TÉCNICAS AVANZADAS DE PREDICCIÓN PARA BIG DATA EN EL CONTEXTO DE SMART CITIES

Tesis doctoral por compendio de artículos

Ricardo - León Talavera Llames

Área de Lenguajes y Sistemas Informáticos.
Escuela Politécnica Superior.
Universidad Pablo de Olavide, Sevilla.

Sevilla. Diciembre de 2018

Documento maquetado con TEXIS v.1.0+.

Este documento está preparado para ser impreso a doble cara.

TÉCNICAS AVANZADAS DE PREDICCIÓN PARA BIG DATA EN EL CONTEXTO DE SMART CITIES

Directores de la tesis
Dr. Alicia Troncoso Lora
Dr. Francisco Martínez Álvarez

Versión 1.0+

Área de Lenguajes y Sistemas Informáticos.
Escuela Politécnica Superior.
Universidad Pablo de Olavide, Sevilla.

Sevilla. Diciembre de 2018

A Martín, y a las
mujeres de mi vida:
mi madre, Yadi y Olivia.

A nosotros nos toca decidir
qué hacer con el tiempo
que se nos ha concedido.
Gandalf el Gris
(El señor de los anillos).

Tesis doctoral subvencionada por la Junta de Andalucía en el proyecto PRY153/14 y por el Data Science & Big Data Research Lab.

Agradecimientos

Pensó que si alguien le había ayudado
tenía que sentirse agradecida y decírselo.
Murmuró: gracias.

Gonzalo Torrente Ballester.

Comenzaré este apartado dando las gracias a Alicia, por la gran oportunidad que me ha brindado al poder realizar esta tesis doctoral, por su paciencia conmigo y, a su vez, por darme la posibilidad de trabajar a su lado. He aprendido mucho con ella y de ella durante estos años.

A mis compañeros del laboratorio, Rubén, Antonio G., Pepe, David, Antonio F. y Samuel, mi pequeña familia en la universidad. Porque han sido cómplices de los éxitos alcanzados y han compartido también conmigo las partes amargas que suponen trabajar en una tesis doctoral. Siempre me han ofrecido un hombro en el que apoyarme y me han sacado una sonrisa cuando menos lo esperaba.

A Gualberto, otra de las personas de las que más he aprendido. Me ofreció su ayuda en un momento difícil, lo que me permitió sacar adelante los resultados que expongo en esta tesis y sin pedir nada a cambio.

A María de la Universidad de Sevilla. Por ser la primera persona que, de forma totalmente desinteresada, me arrojó algo de luz sobre Scala y Spark, y sin cuya guía en los comienzos de esta tesis no me hubiera hecho llegar a este momento. Los algoritmos aquí presentados aún conservan parte de sus enseñanzas.

Al resto de compañeros de la Universidad de Sevilla, José Mari, Fábregas y Álvaro. Porque ellos son el ejemplo de que dos universidades no sólo pueden colaborar y trabajar juntas sino que además pueden alcanzar grandes metas y, al mismo tiempo, crear lazos de amistad que perduren siempre.

A los profesores de la Escuela Politécnica Superior de la Universidad Pablo de Olavide, la mayoría antiguos profesores míos y compañeros durante la realización de esta tesis. Han sido fuente de inspiración y su ejemplo y apoyo me han ayudado mucho en estos años.

A mi familia, especialmente a mi madre. Porque no les he dedicado durante estos años el tiempo que se merecían y necesitaban para dárselo a esta

tesis. Y, a pesar de todo, han sido la mano a la que agarrarme durante las veces que me caía. Son la parte más importante en mi vida y esta tesis es, en gran medida, suya.

A Olivia, mi novia y lo mejor que he me ha pasado en la vida. Comenzamos a andar juntos cuando empezaba esta tesis y ya mi camino lo marca donde ella esté. Ha compartido conmigo cada éxito y cada sinsabor de este trabajo. No hubiera podido llegar hasta aquí si ella no hubiera aparecido en mi vida. Cada sacrificio realizado también ha sido suyo, por lo que esta tesis también es en gran parte suya.

A Pepa, mi coach. Porque me enseñó a organizarme y a buscar la motivación cuando parecía que ya no la había. Su guía y seguimiento han hecho que continuara avanzando con paso firme y me han hecho ser mejor profesional y persona.

A mis hermanos de Erasmus. Porque son los que me han dado vida durante estos años y sólo quedar y vernos me hacía recargar fuerzas para continuar a la vez que olvidarme de mis problemas y preocupaciones. Les debo mucho más de lo que imaginan. Kiitos paljon.

Al resto de mis amigos, otros a los que les he restado tiempo estos años para sacar adelante esta tesis. Y, aun así, siempre me han apoyado para que siguiera y lo han entendido sin que nuestra amistad se resintiera. Las pocas veces que los he visto significaban mucho para mí y me ofrecían ese soplo de aire fresco que necesitaba. Otra parte de esta tesis es de ellos.

Y por último, a Paco, porque necesita una mención especial en esta tesis. Una llamada suya cuando aún estaba en Madrid hizo que comenzara esta aventura. Ha sido mi profesor, mi tutor, mi jefe, pero lo más importante, ha sido y será siempre mi amigo. No tengo palabras para agradecerle todo lo que me ha dado. Siempre dispuesto a ayudar y a sentarse a mi lado cuando lo necesitaba, su paciencia conmigo no tiene recompensa. Gracias por darme esta oportunidad y gracias por ser mi amigo.

Resumen

Todo el mundo trata de realizar algo
grande, sin darse cuenta de que la vida
se compone de cosas pequeñas.

Frank Clark.

Cada día se recoge más y más información de cualquier ámbito de nuestra vida. Número de pasos por minuto, contaminación en las principales ciudades del mundo o el consumo eléctrico medido cada cierto tiempo son sólo algunos ejemplos. Es en este ámbito donde surgen las Smart Cities, o ciudades conectadas, donde se recaba toda la información posible de diferentes dispositivos IoT repartidos por la misma con la esperanza de descubrir conocimiento en dichos datos e, incluso, predecir ciertos comportamientos futuros. Pero estas nuevas series temporales que se están creando comienzan a exceder los tamaños hasta ahora tenidos en cuenta, empezando a considerarse por tanto Big Data.

Las técnicas de machine learning y minería de datos que hasta ahora ofrecían buenos resultados, no podían gestionar tal cantidad de información. Es por ello que necesitaban ser revisadas. Así, surge este trabajo de investigación, donde se propone un algoritmo de predicción basado en vecinos cercanos, para predecir series temporales Big Data. Para ello, apoyándose en nuevos frameworks de análisis de datos como Apache Spark con la computación distribuida como bandera, se proponen dos algoritmos: uno basado en el kWNN para análisis y predicción de series temporales univariante y el MV-kWNN en su versión multivariante.

Se detalla en este trabajo los pasos realizados para adaptarlo a la computación distribuida y los resultados obtenidos tras llevar a cabo la predicción sobre los datos de consumo eléctrico de 3 edificios de una universidad pública. Se muestra, así mismo, las mejoras introducidas al algoritmo para seleccionar de forma óptima los parámetros requeridos por el mismo, estos son: el número de valores pasados que hay que usar (w) para predecir los h valores siguientes y el número de vecinos cercanos k a considerar para la predicción. También se valoran diferentes tamaños de horizontes de predicción h como dato de entrada al algoritmo. Se comprueba la validez de dichas mejoras

realizando la predicción sobre una serie temporal el doble de grande que la considerada en primer término, en este caso la demanda eléctrica en España recogida durante 9 años. Las baja tasa de error obtenida demuestra la idoneidad del algoritmo, y su comparación con otros métodos como deep learning o árboles de regresión, así lo reafirman. Distintas pruebas sobre la escalabilidad del algoritmo en un clúster con diferentes configuraciones muestran lo importante que es escoger adecuadamente parámetros como el número de cores a utilizar por máquina, el número de particiones en que dividir el conjunto de datos así como el número de máquinas en un clúster.

Para finalizar, se propone un nuevo algoritmo para tener en cuenta no sólo una variable, sino varias series exógenas que pudieran mejorar la predicción final. Llevando a cabo diferentes análisis basados en correlación, se define el grado mínimo que deben cumplir las series para mejorar dicha predicción. Experimentaciones sobre dos series reales, de demanda eléctrica en España y del precio de la electricidad durante el mismo periodo, son llevadas a cabo, alcanzando de nuevo bajas tasas de error. La comparación con otros métodos multivariantes, como los de redes neuronales o random forests, sitúan al método propuesto en el primer lugar por delante de estos. Una última experimentación para confirmar la adecuación del algoritmo a series temporales Big Data es realizada, mostrando los tiempos de ejecución multiplicando hasta por 200 el tamaño original de las series.

Índice

Índice de figuras

Índice de Tablas

Parte I

Memoria

Capítulo 1

Introducción

Sólo hay una manera de llegar al destino: comenzar.

Sri Chinmoy (*Setenta y siete mil árboles de servicio*).

Resumen: En este capítulo se plantea el problema que ha intentado resolver esta tesis con los trabajos publicados. Así mismo, se enumeran y describen los objetivos que se marcaron alcanzar y cubrir durante esta investigación. Para finalizar, se detalla y resume la estructura de la memoria de la tesis.

1.1. Planteamiento del problema

Scientia potentia est, o dicho en otras palabras, *El conocimiento es poder*. Desde el comienzo de la humanidad esta frase ha sido frecuentemente utilizada. El ser humano es el único animal racional que existe en la naturaleza y, como tal, hay un aspecto que siempre le ha atraído por encima de cualquier otro: la búsqueda de conocimiento, al considerarse éste como una poderosa herramienta para conseguir sabiduría y riqueza. Por lo tanto, ésta ha sido siempre su meta, y no se ha parado ante nada para conseguirlo. Ni siquiera ante su propia antítesis, el desconocimiento. Es algo que se encuentra en la naturaleza de todos y cada uno de los seres humanos.

Pero, ¿cómo extraer dicho conocimiento? Es más, ¿dónde puede encontrarse? En épocas pasadas se consideraba que todo el conocimiento se podía encontrar estudiando o leyendo. Sin embargo, en la sociedad de la información en la que nos encontramos, donde cada dato de nuestra vida diaria es registrado y organizado, es en el análisis de dichos datos donde podemos hallar ese conocimiento.

Surgieron entonces técnicas y algoritmos para tratar los datos y extraer información de ellos mediante lo que conocemos hoy día como minería de datos y machine learning. Ambas disciplinas están muy relacionadas entre sí y muchas veces se hace complicado separar sus contextos.

Podríamos definir machine learning como el conjunto de técnicas que permiten a un programa *aprender* basándose en los datos que analiza, mejorando así su rendimiento en una tarea específica [8]. El término fue usado por primera vez por A. Samuel en 1959 para definir un videojuego de las damas que él mismo desarrolló y que es considerado como el primer programa con aprendizaje automático [73].

Cuantos más datos se iban generando y recopilando, más difícil se hacía el tratamiento y extracción de información de los mismos, ya que dicho volumen excede nuestra habilidad para reducirlos y analizarlos sin el uso de técnicas de análisis automatizadas. Es en este contexto donde surge entonces la minería de datos, que no es más que una de las etapas más importantes del descubrimiento de la información en bases de datos (KDD o *Knowledge discover in databases*) [24]. Se podría definir la minería de datos como el proceso de descubrimiento de patrones en repositorios de datos de gran tamaño usando para ello técnicas de machine learning, técnicas estadísticas y técnicas de sistemas de base datos [39]. Definido en varias fases, este proceso se puede entender entonces como el proceso completo de extracción de información, que se encarga así mismo de la preparación de los datos y de la interpretación de los resultados obtenidos. Cabe mencionar en este apartado que algunos autores distinguen dos tipos de minería de datos [24]:

1. Minería de datos predictiva. En otras palabras, predicción de datos, basándose en técnicas estadísticas. La clasificación y la regresión son las tareas sobre datos que producen modelos predictivos.

    - Clasificación. Es la técnica más usada. Se considera que cada registro de un conjunto de datos pertenece a una determinada clase o etiqueta discreta, que se indica mediante el valor de un atributo o clase de la instancia. El objetivo no es otro que predecir una clase, dados los valores de los atributos. Árboles de decisión [70] o análisis discriminante [55] son algunos ejemplos.
    - Regresión o estimación. Es el aprendizaje de una función real que asigna a cada instancia un valor real de tipo numérico. El objetivo es inducir un modelo para poder predecir el valor de la clase dados los valores de los atributos. Las técnicas más conocidas son redes neuronales [93] y regresión lineal [96].

2. Minería de datos para el descubrimiento de conocimiento. Tareas que producen modelos descriptivos. Las más conocidas son el agrupamiento (clustering), las reglas de asociación secuenciales y el análisis correlacional.

    - Clustering [23]. Consiste en la obtención de grupos, que tienen elementos similares, a partir de los datos. Estos elementos u objetos similares de un grupo son muy diferentes a los objetos de otro grupo.
    - Reglas de asociación [1]. Su objetivo es identificar relaciones no explícitas entre atributos categóricos o numéricos. Una de las variantes de reglas de asociación es la secuencial, que, como su propio nombre indica, usa secuencias de datos.
    - Análisis correlacional [37]. Utilizada para comprobar el grado de similitud de los valores de dos variables numéricas.

Como se ha comentado anteriormente, las técnicas actuales de minería de datos y machine learning venían ofreciendo buenos resultados cuando la cantidad y volumen de datos era limitada. Pero, ¿respondían de igual manera dichas técnicas cuando la información empezaba a ser considerable?

Gracias a los últimos avances tecnológicos realizados hasta la fecha, almacenar grandes cantidades de información ya no supone ningún inconveniente. Es más, el precio por el almacenamiento de datos se ha reducido y el acceso a ordenadores de alto rendimiento se ha hecho más asequible, en gran medida por la irrupción de la computación en la nube, más conocida por su acepción inglesa *cloud computing* (Amazon Web Services [74], Google Cloud Storage [79] u Open Telekom Cloud [18] son sólo algunos ejemplos). De hecho se estima que el 90 % de los datos actualmente registrados han sido generados sólo en la última década. La EMC Digital Universe study, sociedad encargada de cuantificar y predecir la cantidad de datos producidos anualmente, ha ido más allá y en 2014 estableció que para el año 2020 se alcanzarían los 44 zettabytes ($4{,}4 \times 10^9$ terabytes) de datos almacenados, lo que suponía diez veces más que la cantidad existente hasta ese momento [80]. La gestión de tal cantidad de datos y el análisis de los mismos dio pie a lo que hoy en día denominamos Big Data [51]. Una definición para el término Big Data podría ser el dado en [53]: *represents the information assets characterized by such a high volume, velocity and variety to require specific technology and analytical methods for its transformation into value.* Podemos extraer de dicha definición los términos más importantes, como son *volumen, velocidad, variedad* y, la más importante, *valor* entendiendo este último como información de interés o conocimiento. Uno de los campos que ha contribuido de manera notable al crecimiento de dicha cantidad de información, y en definitiva al Big Data, ha sido el Internet de las Cosas, o en su acepción inglesa *Internet of Things* (IoT).

El término IoT fue utilizado por primera vez en [7], y aunque no existe una definición estándar y exacta del mismo, podemos tomar las siguientes como las más idóneas:

1. El origen semántico de la expresión está compuesto por dos conceptos: *Internet* y *Thing*. *Internet* se puede definir como la red mundial de redes de ordenadores interconectados, basado en un protocolo estándar de comunicaciones (TCP/IP). *Thing* se puede definir como un objeto no identificable de forma precisa. De este modo, semánticamente IoT se puede definir como una red mundial de objetos interconectados, con direcciones únicas y basado en un protocolo estándar de comunicaciones [10].

2. IoT permite a las personas y a los objetos estar conectados a todas horas, en cualquier lugar, mediante cualquier dispositivo y con cualquier persona usando para ello cualquier ruta o red y cualquier servicio [35].

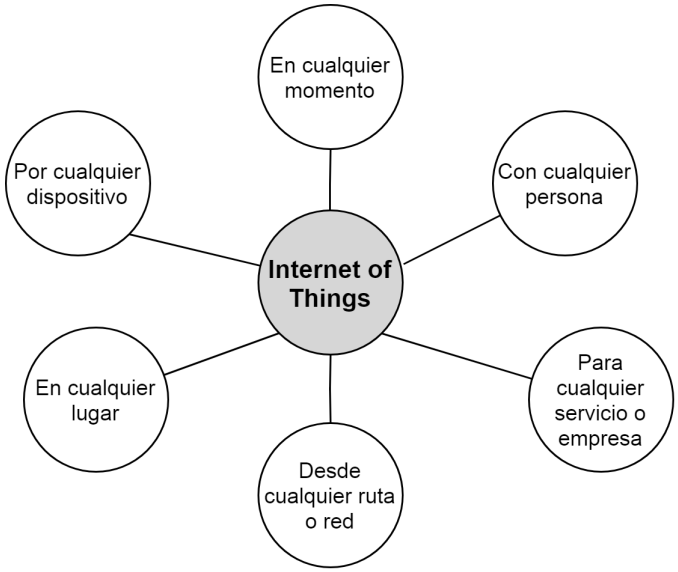


Figura 1.1: Definición más consensuada para IoT.

En pocas palabras, cualquier dispositivo puede convertirse en un sensor IoT que recopile medidas y datos y las transmita a internet sin que medie interacción humana. Gobiernos y toda clase de empresas han tomado como punto de partida el IoT para *sensorizar* y recopilar la mayor cantidad de datos posible de todo tipo de infraestructuras y, en general, de ciudades enteras, dando lugar a lo que conocemos como *Smart Cities*. La recopilación de información en una Smart City tiene como principales objetivo optimizar la eficiencia en las operaciones y servicios y conectar con los ciudadanos [65].

De esta forma, el análisis de la información emitida por un sólo dispositivo midiendo un par de parámetros periódicamente se puede realizar mediante técnicas estadísticas clásicas [91] o de machine learning. Pero el número de dispositivos IoT en una Smart City es considerable, y más aún la cantidad de datos recopilados por todos ellos.

Debido a todo lo anteriormente expuesto, a medida que el tamaño de los datos almacenados ha ido aumentando, la eficiencia y eficacia de dichas técnicas de machine learning han ido descendiendo. El estudio y realización de este trabajo de investigación que se expone ha intentado abordar la revisión de estas técnicas para adaptarlas al ámbito de Big Data, en el contexto de Smart Cities, ofreciendo resultados prometedores al respecto.

Pero, ¿y si además de extraer valor o conocimiento de los datos pudiéramos también aprovecharlos y dar un paso más, e intentar adelantar eventos futuros? Aquí se encuentra otra de las obsesiones de cualquier persona, el conocimiento del futuro. Anticipar acontecimientos y actuar en consecuencia ha sido el objetivo de muchos científicos a lo largo de toda la historia. El ser humano siempre ha querido conocer el futuro. Sobre todo para actuar en base a ello y antes de que los hechos así se dieran. En el contexto de Smart Cities, predecir los atascos de tráfico [6], el consumo eléctrico [63] o la polución en la grandes ciudades [95] puede ayudar a mejorar la vida en dichas poblaciones además de ahorrar grandes cantidades de dinero.

En conclusión, este trabajo de investigación ha tenido como objetivo adaptar un algoritmo de clasificación de machine learning, el algoritmo de vecinos cercanos, al contexto de Big Data y además mejorarlo para usarlo en problemas de predicción, implementándolo en dos modalidades: una versión univariante y otra versión multivariante, para aplicarlo a datos energéticos reales producidos en una Smart City.

1.2. Objetivos de esta tesis

El objetivo fundamental que persigue esta tesis doctoral es la predicción de series temporales en un entorno Big Data y su aplicación en el contexto de Smart Cities. Para ello, se ha desarrollado un algoritmo de predicción, basado en el algoritmo de vecinos cercanos (kNN), tanto en su versión univariante como en su versión multivariante, que pueda aplicarse a cualquier conjunto de datos temporales de tales dimensiones que pueda ser considerado Big Data. Este objetivo se puede dividir en los subobjetivos que se enumeran a continuación:

1. Desarrollo de algoritmo kNN para Big Data: el algoritmo kNN es muy costoso computacionalmente ya que calcula todas las distancias existentes entre las instancias que forman el conjunto de datos. Si n es el número total de instancias, se calculan $n(n-1)/2$ distancias (proporcional a n^2). Por tanto, en conjuntos de datos masivos en un entorno

Big Data, es necesario desarrollar el algoritmo kNN para su ejecución en una plataforma de computación distribuida. De esta forma, el primer subobjetivo ha sido la implementación del algoritmo kNN para poder ser usado en un contexto Big Data. En concreto, se ha usado la plataforma Apache Spark para realizar dicha implementación.

2. Diseño y desarrollo de algoritmo de predicción univariante basado en kNN: una vez implementado el algoritmo kNN para Big Data, se ha pretendido desarrollar una técnica de predicción, basada en dicho algoritmo, especialmente diseñada para series temporales de gran tamaño, en las que las técnicas de predicción existentes en la literatura no pueden ser aplicadas.

3. Diseño y desarrollo de algoritmo de predicción multivariante basado en kNN: habiendo implementado el algoritmo de predicción univariante, se ha desarrollado a su vez su versión multivariante para procesar más de una serie temporal al mismo tiempo, analizar su relación, predecir todas ellas e intentar mejorar los posibles resultados que se obtuvieran con el algoritmo univariante.

4. Resultados experimentales: aunque, como se comenta anteriormente, se persigue que la técnica de predicción pueda ser aplicada a cualquier conjunto de datos temporales de grandes dimensiones, se ha hecho especial hincapié en su validación sobre problemas reales en el contexto de Smart Cities. Para ello, los conjuntos de datos usados por el algoritmo desarrollado han sido el consumo eléctrico medido cada 15 minutos durante 3 años en varios edificios de la Universidad Pablo de Olavide, el consumo eléctrico en España medido cada 10 minutos durante 9 años y el precio de la electricidad medido cada hora durante los mismos 9 años.

1.3. Principales contribuciones de la tesis doctoral

En esta sección se detallan las principales contribuciones alcanzadas durante la tesis doctoral y en qué artículos pueden encontrarse.

[83] R. L. Talavera-Llames, R. Pérez-Chacón, M. Martínez-Ballesteros, A. Troncoso, and F. Martínez-Álvarez. *A nearest neighbours-based algorithm for big time series data forecasting.* In International Conference on Hybrid Artificial Intelligence Systems, páginas 174-185, 2016.

En este trabajo de investigación se presentó un algoritmo de predicción para Big Data basado en vecinos cercanos. El algoritmo se desarrolló en el framework de Apache Spark implementándose en el lenguaje Scala. El

algoritmo fue probado sobre un conjunto de datos de consumo eléctrico, recogidos por una red de sensores, en dos edificios de una universidad pública, obteniéndose resultados prometedores.

[63] R. Pérez-Chacón, R. L. Talavera-Llames, F. Martínez-Álvarez, and A. Troncoso. *Finding electric energy consumption patterns in big time series data.* In Proceedings of the 13th International Conference on Distributed Computing and Artificial Intelligence, páginas 231-238, 2016.

En este trabajo se presenta un algoritmo de clustering desarrollado también en Apache Spark. Se trata del algoritmo k-means y su objetivo era encontrar patrones en series temporales Big Data y, a su vez, predecir sus valores futuros. Se utilizaron para ello series temporales de consumo eléctrico en varios edificios de una universidad pública. El algoritmo es comparado, en tiempos de ejecución, con Weka multiplicando el tamaño de los datos hasta por 300. Destacar que los resultados de este trabajo preliminar motivaron el algoritmo de predicción de series temporales publicado en [83].

[81] R. L. Talavera-Llames, R. Pérez-Chacón, A. Troncoso, and F. Martínez-Álvarez. *Big data time series forecasting based on nearest neighbours distributed computing with Spark.* Knowledge-Based Systems, volumen 161, páginas 12-25, 2018.

En este trabajo de investigación se mejoró el algoritmo basado en vecinos cercanos para predicción desarrollado en [83], haciendo además que la selección de parámetros se realizara de forma automática eligiéndose siempre los que fueran más óptimos. De esta forma, se podían procesar de forma aún más rápida series temporales de grandes dimensiones debido a la computación distribuida en la que se basaba. Se realizaron experimentaciones tomando como datos de entrada la demanda eléctrica medida cada 10 minutos durante más de 9 años, obteniéndose un error en la predicción de 1.63 %, demostrando la robustez del algoritmo. Además, se compararon los errores en la predicción con los resultados obtenidos con nuevas técnicas en entornos Big Data, tales como deep learning, árboles de regresión, random forests y Gradient-Boosted Trees, usando para ello el mismo conjunto de datos, mejorando a dichas técnicas en un 39.68 % de media. Para finalizar, se realizó un estudio de escalabilidad del algoritmo propuesto, con el objetivo de obtener los parámetros óptimos de configuración en un clúster. Los parámetros considerados fueron el número de cores a utilizar en cada máquina del clúster, el número de particiones en el que dividir el conjunto de datos de entrada y el número de nodos en un clúster. Se demostró que 16 esclavos, con 8 cores por máquina y 72 particiones del conjunto de datos eran los valores óptimos a tomar para la serie temporal en estudio.

[28] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, and F. Martínez-Álvarez. *Multi-step forecasting for big data time series based on ensemble learning*. Knowledge-Based Systems, volumen 163, páginas 830-841, 2019.

En este trabajo de investigación se implementó un nuevo método, desarrollado así mismo en Apache Spark, y que combinaba random forests, árboles de regresión y Gradiented-Boosted Trees para predecir series temporales de gran tamaño. Así, el algoritmo, siguiendo la técnica de mínimos cuadrados, otorgaba un determinado peso a cada método según la precisión en su predicción. La manera de actualizar los pesos, ya sea dinámica o estáticamente, es analizado en el estudio. Utilizando dos series temporales de electricidad, se analiza la precisión en la predicción del algoritmo comparándolo con los métodos por sí solos y con redes neuronales, PSF y deep learning, mejorándolos a todos. Destacar que en este estudio se desarrollaron los algoritmos que formaron parte del bechmark de comparación para validar el algoritmo de predicción univariante desarrollado en [81].

[82] R. L. Talavera-Llames, R. Pérez-Chacón, A. Troncoso, and F. Martínez-Álvarez. *MV-kWNN: A novel multivariate and multi-output weighted nearest neighbours algorithm for big data time series forecasting*. Neurocomputing, in press, 2018.

En este trabajo de investigación se propuso un nuevo algoritmo de predicción, basado también en vecinos cercanos, para que tuviera en cuenta más de una variable y fuera capaz de predecir todas ellas. El algoritmo también fue desarrollado en el framework Apache Spark para, así, poder llevar a cabo predicciones con horizontes de predicción arbitrarios sobre datos de gran tamaño. Se utilizaron para las experimentaciones datos de los mercados eléctricos en España, concretamente la demanda eléctrica medida cada 10 minutos y los precios de la electricidad medidos cada hora durante más de 9 años. Los resultados obtenidos demostraron que conjuntos de datos de millones de registros podían ser procesados de forma eficiente por el algoritmo. La relación lineal entre el tamaño del conjunto de datos y los tiempos de ejecución obtenidos confirmaron la idoneidad del método dado la escalabilidad que éste tenía al procesar series temporales Big Data. Se generaron conjuntos de datos sintéticos para analizar empíricamente la funcionalidad del algoritmo, descubriendo el umbral mínimo de correlación que una variable exógena debía cumplir respecto a una variable objetivo para mejorar los resultados de una predicción. Además, se compararon los resultados obtenidos por el algoritmo propuesto con versiones multivariantes de random forests, redes neuronales y de modelos clásicos de Box y Jenkins tales como modelos autorregresivos (ARX), modelos autorregresivos de medias móviles (ARMAX) y modelos autorregresivos integrados de media móvil (ARIMAX). Se confirmó que el algoritmo propuesto mejoraba a todos ellos, en términos de precisión

en la predicción y demostrándose que los métodos actuales multivariantes no eran capaces de procesar grandes tamaños de información, como sí hacia el método propuesto. Finalmente, se realizaron diversos tests estadísticos para probar la validez del método sobre el resto de los algoritmos considerados.

1.4. Estructura de la memoria de la tesis doctoral

En está sección se detalla la estructura que sigue esta tesis doctoral y las partes que la componen.

- La Parte I se divide en 5 capítulos:
  - El Capítulo 1 introduce el problema abordado por la tesis, los objetivos que se perseguían con la realización de la misma, así como las principales contribuciones aportadas, finalizando con la descripción de la memoria de la tesis en sí.
  - El Capítulo 2 plantea y detalla el marco teórico en el que se encuadra el trabajo de esta tesis doctoral. En una primera sección se desarrolla el concepto de serie temporal, analizando sus posibles componentes y comportamientos, se plantea en qué consiste un problema de predicción de una serie temporal y los métodos existentes hasta el momento, continuando analizando las series temporales multivariantes para terminar comentando las técnicas actuales para predecir este tipo de series. En una segunda sección se detalla el problema que supone predecir series temporales Big Data junto con los trabajos publicados al respecto y se describen los nuevos frameworks de procesamiento de Big Data aparecidos hasta la fecha.
  - El Capítulo 3 pormenoriza los detalles metodológicos y de implementación de los algoritmos propuestos en esta tesis doctoral. Se comienza comentando las etapas de desarrollo de un algoritmo de predicción de series temporales Big Data en el framework Apache Spark. A continuación se describen las adaptaciones y mejoras introducidas al algoritmo para ofrecer mejores resultados y se finaliza detallando la implementación llevada a cabo para desarrollar un algoritmo de predicción multivariante.
  - El Capítulo 4 plantea la discusión conjunta de los resultados obtenidos en esta investigación tras aplicar los algoritmos a diferentes conjuntos de datos. En una primera sección se comentan las principales conclusiones obtenidas tras utilizar el algoritmo univariante sobre el consumo energético en edificios de una universidad pública. En una segunda sección de detallan los resultados obtenidos después de aplicar los algoritmos, tanto el univariante como el multivariante, sobre la demanda de energía eléctrica en España.

- El Capítulo 5 recoge las principales conclusiones extraídas tras realizar este trabajo de investigación, las cuales llevan además a considerar posibles alternativas como trabajos futuros a desarrollar.

- La Parte II recoge los trabajos de investigación publicados durante la realización de esta tesis doctoral, tanto los que se consideran para el compendio de artículos como aquellos en los que también se ha trabajado y han servido de apoyo al estudio de investigación expuesto.

Capítulo 2

Marco teórico

La mejor forma de predecir el futuro es implementarlo.

David Heinemeier Hansson (creador de Ruby on Rails).

Resumen: En este capítulo se detalla el marco teórico en el que se encuadra el conjunto de publicaciones. En una primera parte se analizan las series temporales, centrándose en la definición de serie temporal como tal junto con sus componentes y comportamiento. También se pormenoriza en qué consiste predecir series temporales y los métodos utilizados hasta el momento, pasando a analizar las series temporales multivariantes así como los procedimientos empleados para predecirlas. En una segunda parte se estudia la predicción de series temporales Big Data y algunos trabajos importantes publicados hasta el momento junto con el análisis de los nuevos frameworks de procesamiento que han permitido tratar tal tamaño de datos.

2.1. Series temporales

2.1.1. ¿Qué es una serie temporal?

Cualquier valor medido a lo largo del tiempo durante un periodo determinado podría considerarse una serie temporal. Aunque una descripción más exacta podría definir una serie temporal como las medidas de una variable indexadas cronológicamente o bien como una secuencia de datos tomados en distintos instantes de tiempo. De esta forma, podemos encontrar series temporales en cualquier ámbito de la vida cotidiana.

Algunos ejemplos podrían ser:

- En meteorología: evolución de las temperaturas mínimas o precipitaciones diarias.
- En economía: índice del precio del petróleo o cambios en el precio del alquiler.
- En medio ambiente: frecuencia de terremotos o evolución de las emisiones de CO_2.
- En demografía: tasa de nacimientos o crecimiento/decrecimiento del número de habitantes.
- En medicina: latidos del corazón por minuto o evolución temporal de un virus tras la aplicación de un nuevo fármaco.
- En agricultura: producción anual de cultivos o producción de ganado.

La lista de áreas en las que se estudia las series temporales es ciertamente elevada. El objetivo de analizarlas es, por un lado, entender o modelar el mecanismo que da lugar a una serie observada y por otro predecir los futuros valores de la serie basándose en el pasado de dicha serie o en otras series o factores relacionados.

Pero no todas las series temporales se comportan de igual forma en el tiempo, lo que lleva a poder clasificarlas de la siguiente forma:

- Estacionarias: se definen como series temporales cuya media y variabilidad permanece constante en el tiempo, es decir, series que se comportan con cierta estabilidad. Las medidas en este tipo de series no presentan aumentos o disminuciones muy pronunciadas, fluctuando sus valores en torno a la media y manteniéndose la variabilidad de su valores constante.
- No estacionarias: se definen como series temporales cuya media y/o variabilidad no permanecen constantes sino que van oscilando con el tiempo. De esta forma, estas series suelen mostrar una tendencia a decrecer o crecer a lo largo del tiempo cuando se producen cambios en la media. Así mismo, pueden detectarse comportamiento similares en los valores de la serie en momentos periódicos de tiempo, reflejando efectos estacionales.

De este modo, una serie temporal estacionaria será más fácil de predecir que una no estacionaria, ya que ésta última requerirá de un mayor conocimiento y análisis para estimar valores futuros. Es por ello que resulta de vital importancia conocer y estudiar los componentes de una serie temporal.

Estos son:

- Tendencia: define a largo plazo un posible cambio en la serie, bien porque se produce una variación en torno al nivel medio o bien porque se produce una variación de la media a largo plazo. La tendencia se suele manifestar en una serie como un movimiento o comportamiento suave a largo plazo.
- Estacionalidad: se corresponde con ciertos movimientos periódicos de oscilación de los valores durante el tiempo que comprende la serie. Son fáciles de interpretar y analizar.
- Irregular o componente aleatoria: se corresponde con el resto de valores, que pueden considerarse aleatorios, con un comportamiento variable.

Así, una serie temporal se puede describir en función de los componentes anteriores:

$$V(t) = f(T(t), E(t), A(t)) \tag{2.1}$$

siendo V(t) la serie temporal, T(t) la tendencia de la serie, E(t) su estacionalidad y A(t) la componente aleatoria.

Existen varias maneras de representar la Ecuación (2.1). La forma exacta dependerá del método de descomposición utilizado. La aproximación más común suele ser la suma de los componentes, es decir, que simplemente la agregación de la tendencia, la estacionalidad y la componente aleatoria darían como resultado la serie observada, presentándose de la siguiente forma:

$$V(t) = T(t) + E(t) + A(t) \tag{2.2}$$

Otra forma comúnmente aceptada es la descomposición multiplicativa, siendo la forma:

$$V(t) = T(t) \times E(t) \times A(t) \tag{2.3}$$

donde la tendencia, la estacionalidad y la componente aleatoria se multiplican entre ellas para dar la serie temporal observada.

Un modelo aditivo se caracteriza porque los efectos de sus componentes individuales son tanto agregados como diferenciados de forma conjunta para que así los datos sean modelados, por lo que resulta apropiado si la magnitud de las fluctuaciones de la estacionalidad no varía con el nivel de la serie. Sin embargo, si dichas fluctuaciones se incrementan o decrementan proporcionalmente con incrementos o decrementos de la serie, entonces un modelo multiplicativo sería el más adecuado. Aunque también se puede utilizar una transformación, en lugar de escoger entre una descomposición aditiva o multiplicativa. Muy frecuentemente la serie transformada se puede modelar de forma aditiva aunque los datos no lo sean.

Los logaritmos, por ejemplo, transforman una relación multiplicativa en una relación aditiva ya que:

$$\begin{aligned} V(t) &= T(t) \times E(t) \times A(t) \\ \log V(t) &= \log T(t) + \log E(t) + \log A(t) \end{aligned} \tag{2.4}$$

De esta forma se puede ajustar una relación multiplicativa llevando a cabo una transformación a una relación aditiva de los logaritmos de los datos.

Existen además otras transformaciones que permiten una descomposición que se podrían situar entre las formas aditivas y las multiplicativas. Un ejemplo de éstas sería la descomposición pseudo-aditiva que se representa como:

$$V(t) = T(t)(E(t) + A(t) - 1) \tag{2.5}$$

Resulta un tipo de descomposición de gran utilidad en series temporales donde existe un mes cuyos valores son más altos o más bajos que el resto de meses. Podría ser el caso de grandes bajadas en series temporales eléctricas en algunos países de Europa, donde las empresas cierran por vacaciones.

Es importante tener en cuenta todos estos modelos de descomposición de una serie temporal, ya que permiten establecer un modelo matemático de la serie que ayudará a entender mejor la misma y, por ende, ayudará a predecir mejores valores.

Si nos centramos en el análisis de la componente de la tendencia, se podría nombrar a la serie temporal por Z_t, dependiendo sólo del índice t, que se corresponde con el periodo de tiempo principal, ya que el objetivo no es otro que aislar el movimiento a largo plazo de la serie. Para ello existen diferentes métodos que se pueden utilizar [64]:

- Método de ajuste analítico. Se realiza un ajuste por regresión de los valores de la serie a una función del tiempo que sea sencilla y a su vez que recoja de forma satisfactoria la marcha general del proceso representado por la serie temporal. Pueden realizarse ajustes a tendencias de diversos tipos aunque las más comunes son:

  - Tendencia lineal. Consiste en una línea recta que se ajusta correctamente a los datos. Indica que algo aumenta o disminuye a un ritmo constante, siguiendo la fórmula:

    $$Z_t = a + bt \tag{2.6}$$

  - Tendencia logarítmica. Es una línea curva cuando el índice de cambios de los datos aumenta o disminuye rápidamente y, después, se estabiliza. Puede utilizar valores positivos o negativos, y sigue la fórmula:

    $$Z_t = \log(a + bt) \tag{2.7}$$

- Tendencia polinómica. Consiste en una línea curva que se utiliza cuando los datos fluctúan según la ecuación de un polinomio. El orden del polinomio se determina mediante el número de fluctuaciones de los datos o en función del número de máximos y mínimos que aparecen en la curva. Así, una tendencia polinómica de orden 2 podrá tener un máximo o un mínimo, una de orden 3 tendrá uno o dos máximos o mínimos, mientras que una de orden 4 tendrá más de tres. Suele utilizarse para analizar las pérdidas y ganancias de un conjunto de datos grande. Sigue la fórmula:

$$Z_t = a + bt + ct^2 + ... + ct^n \tag{2.8}$$

- Tendencia potencial. Se trata de una línea curva que se utiliza con conjuntos de datos que comparan medidas que aumentan a un ritmo concreto. No es posible crear una línea de tendencia potencial si los datos contienen valores ceros o negativos. Se representa por la fórmula:

$$Z_t = at^b \tag{2.9}$$

- Tendencia exponencial. Es una línea curva que resulta de gran utilidad cuando los valores de los datos aumentan o disminuyen a intervalos cada vez mayores. Como en la tendencia anterior, no se podrá crear una línea de tendencia exponencial si los datos contienen valores ceros o negativos. Sigue la fórmula:

$$Z_t = exp(a + bt) \tag{2.10}$$

- Tendencia de media móvil. Consiste en una línea que atenúa las fluctuaciones en los datos para mostrar con mayor claridad la trama o la tendencia, para ello utiliza un número concreto de observaciones de los datos (se establecen por la opción Período), hace un promedio de los mismos, y utiliza éste como punto de la línea.

▪ Método de las medias móviles de orden p. Analiza la tendencia de una serie temporal a partir del resumen de los datos iniciales mediante determinadas medias de los mismos. De esta forma, si p es impar, se forman medias relativas a los instantes $(p+1)/2$, $(p+3)/2$, $(p+5)/2$, etc. Dará como resultado valores enteros dado que p es impar. Por otro lado, si p es par, se forman medias relativas a los instantes $(p+1)/2$, $(p+3)/2$, $(p+5)/2$, etc. Dará como resultado valores no enteros dado que p es par. Seguidamente, se hallan nuevas medias móviles entre cada dos medias móviles originales consecutivas, que serán ahora relativas a los instantes $(p+2)/2$, $(p+4)/2$, $(p+6)/2$, etc. Resultando ya en valores enteros dado que p es par. Una vez obtenida la serie de medias móviles, la tendencia será la línea que las une.

- Método de las diferencias. Consiste en derivar de los datos originales y_t un nueva serie z_t que será obtenida como la diferencia entre el valor de la variable en el momento actual y el valor en el momento inmediatamente anterior, resultando en:

$$z_t = y_t - y_{t-1} = \nabla y_t \tag{2.11}$$

  Se comprueba entonces si z_t oscila alrededor de un mismo valor, con lo que indicaría que la serie no tendría tendencia. Sin embargo, si z_t crece o decrece a largo plazo habría que seguir calculando una nueva serie de diferencias w_t definida de la siguiente forma:

$$w_t = z_t - z_{t-1} = \nabla z_t = \nabla\nabla y_t = \nabla^2 y_t \tag{2.12}$$

Si nos centramos en la componente estacional [64], el objetivo que subyace de estudiarla es que en la amplía mayoría de las series temporales provoca una distorsión de su verdadero movimiento. Para eliminar estas distorsiones y captar el movimiento real de la serie, se requiere desestacionalizar la misma.

La desestacionalización puede considerarse como una tarea no trivial o como una tarea trivial. Si la consideramos no trivial podemos destacar los siguientes estudios o algoritmos:

- Census Bureau's X-11 [76]: este programa estima la estacionalidad aplicando filtros de medias móviles a una versión modificada de los datos de entrada de la serie temporal.

- Census Bureau's X-12 [25]: similar al anterior pero ofreciendo diversas mejoras. Entre ellas se pueden incluir ajustes para valores extremos de la serie o ajustes para efectos vacacionales que también son estimados por el programa. Los filtros son elegidos de un subconjunto fijo de estos, posiblemente de forma completamente automática, sobre la base de determinadas relaciones señal/ruido.

Sin embargo, si se considera la desestacionalización como una tarea trivial, se pueden utilizar los siguientes métodos:

- Método de desestacionalización de la tendencia: este método comienza ajustando una recta mediante mínimos cuadrados a las medias anuales de los datos observados. Después se calcula las medias mensuales en los diferentes años. Se aísla entonces la componente estacional obteniendo las medias mensuales corregidas y con ellas se calcula la media global corregida. Si el esquema es multiplicativo, se calculan a su vez los índices de variación estacional y se desestacionaliza la serie dividiendo sus valores por estos índices. Sin embargo, si el esquema es aditivo, la componente estacional de cada mes resultará de extraer a su media mensual corregida la media global corregida.

- Método de desestacionalización del índice estacional: dicho método empieza hallando la tendencia mediante el método de las medias móviles pero tomando solamente un año de período. Centra entonces los valores obtenidos en los instantes de tiempo originales y elimina la tendencia. Seguidamente, elimina las variaciones irregulares y sobre los valores observados en cada período de repetición anual se calculan los índices de variación estacional. Finalmente se dividen los valores de la serie original por estos índices obteniéndose la serie desestacionalizada.

- Método de desestacionalización de las medias móviles: este método consiste en obtener la componente extraestacional mediante un ajuste de la serie original por medias móviles de orden m (período estacional) para eliminar las variaciones estacionales.

- Método de desestacionalización de las diferencias estacionales: dicho método permite eliminar la mayor parte del efecto estacional de una serie, y consiste en obtener la serie de diferencias de orden m (período estacional).

El objetivo de estimar y extraer la tendencia y la estacionalidad (componentes deterministas) reside en la esperanza de que la componente aleatoria se pueda definir como un proceso estacionario aleatorio. Los procesos estacionarios son más fáciles de analizar y predecir ya que son procesos estocásticos cuyos medias y varianzas no dependen del tiempo. De hecho, se podría usar la teoría de tales procesos para encontrar un modelo probabilístico satisfactorio de la componente irregular y, así, analizar sus propiedades y en conjunción con la tendencia y la estacionalidad, usarlos para predecir o controlar la serie temporal.

Otra aproximación alternativa a la anterior sería la desarrollada por Box y Jenkins [11]. Estos investigadores fijaron diversas fases para su modelado como a continuación se detalla:

1. Recogida de datos de la serie: se recomienda disponer de 50 o más datos. Para el caso de series mensuales, se recomienda trabajar con entre seis y diez años completos de información.

2. Representación gráfica de la serie: se realiza como primer paso del proceso de identificación, para así decidir sobre la estacionariedad de la serie.

3. Transformación previa de la serie: dentro del proceso de identificación, la transformación logarítmica será necesaria en caso de serie no estacionaria en varianza. Es una transformación muy frecuente incluso en series con dispersión relativamente constante en el tiempo.

4. Eliminación de la tendencia: habiendo indicado la representación de la serie la existencia o no de tendencia, ésta deberá ser corregida. Una

tendencia lineal se corregirá tomando primeras diferencias, que será el caso más frecuente (d=1). Una tendencia no lineal suele llevar en la práctica al uso de dos diferencias como mucho (d=2). Una vez estacionarizada la serie, se habrá identificado el parámetro d.

5. Identificación efectiva del modelo: en este paso se ha de determinar el tipo de modelo más adecuado para la serie objeto de estudio, esto es, el orden de los procesos autorregresivos p y de medias móviles q de las componentes regular y estacionaria. Técnicamente esta decisión se tomará en base a las funciones de autocorrelación y autocorrelación parcial.

6. Estimación de los coeficientes del modelo: una vez el modelo ha sido elegido, se procede a la estimación de sus parámetros. Al tratarse de un procedimiento iterativo de cálculo, pueden sugerirse valores iniciales.

7. Contraste de validez del modelo o validación: se utilizan diversos procedimientos para valorar el modelo o modelos inicialmente seleccionados, como pudieran ser el contraste de significación de parámetros o las covarianzas entre estimadores.

8. Análisis detallado de los errores: para una valoración final del modelo se realiza una diferencia histórica entre valores reales y estimados por el modelo. De esta forma se comprobará si existe un comportamiento no sistemático de los mismos y se analizará la posible existencia de errores especialmente significativos.

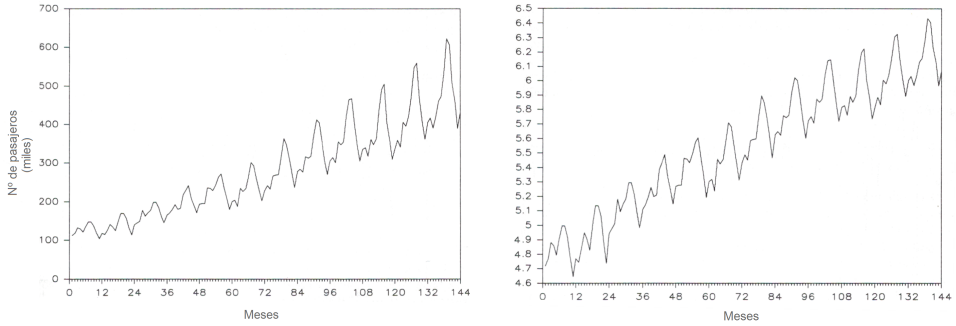
9. Selección del modelo: dependiendo de los resultados obtenidos en los pasos anteriores, se decidirá el modelo más adecuado.

10. Predicción: el modelo seleccionado se usará como fórmula inicial de predicción.

En estos modelos, el término de error es el resultado residual no explicado por el modelo. La suposición de los modelos econométricos es que la varianza de este término será uniforme. Esto es conocido como homocedasticidad. Sin embargo, en algunas circunstancias, esta variación no es uniforme, lo que se denomina como heterocedasticidad. Surgieron entonces nuevos modelos que se utilizaban para analizar estos efectos que no cubrían los modelos econométricos. El modelo autorregresivo con heterocedasticidad o ARCH [21] es uno de ellos. En estos modelos, la varianza de los términos de error no es sólo no uniforme sino que también está afectada por las varianzas que la preceden. Se comprueba entonces que la varianza del error en una serie temporal sigue un modelo autoregresivo (AR). Si la varianza del error sigue un modelo autoregresivo de medias móviles (ARMA), el modelo ARCH se dice que es es generalizado (GARCH). El problema de la heterocedasticidad hace que

los intervalos de confianza sean muy reducidos, ofreciendo de esta forma una mayor sensación de precisión que la que garantiza el modelo econométrico. Los modelos ARCH intentan modelar la varianza de estos términos de error, y, en el proceso, corregir los problemas resultantes de la heterocedasticidad. El objetivo final que persiguen estos modelos es proporcionar una medida de volatilidad que pueda utilizarse en la toma de decisiones financieras.

Sin embargo, después de haber comentado todos los métodos anteriores, es necesario resaltar que la mejor forma de analizar el comportamiento de una serie temporal sigue siendo graficar sus datos. De esta forma, si existieran aparente discontinuidades en la serie, como pudieran ser cambios repentinos de nivel, sería recomendable particionar la serie en segmentos homogéneos para analizarla. Si hubiera valores aislados, deberían ser estudiados cuidadosamente para comprobar si existe alguna justificación para descartarlos, como, por ejemplo, si un valor ha sido registrado por error. Examinando el gráfico además se podría comprobar si las fluctuaciones de los componentes estacionales e irregulares se incrementan con el nivel de la serie, entonces debería llevarse a cabo una primera transformación de los datos para hacer que estos sean compatibles con el modelo. En la Figura 2.1 extraída de [15] se muestra el número de pasajeros de una compañía aérea por meses, desde enero de 1949 hasta diciembre de 1960. La Figura 2.1(a) muestra los valores originales, mientras que la Figura 2.1(b) se muestran los valores obtenidos después de aplicar una transformación logarítmica, como se explicó anteriormente.



(a) Gráfica con los valores originales.

(b) Gráfica después de aplicar una transformación logarítmica.

Figura 2.1: Número de pasajeros de una compañía aérea (1949 - 1960).

2.1.2. Predicción de series temporales

Una vez que una serie temporal se analiza, y lo que es más importante, conocido su comportamiento a través del modelo extraído, se puede llevar a cabo una predicción sobre la misma. Predecir no es más que un intento de conocer el futuro y por tanto, anticiparse a él. En general, los métodos de predicción se pueden dividir en dos bloques: métodos cualitativos y métodos cuantitativos.

Los métodos cualitativos se usan en casos en los que el pasado no aporta información relacionada con el proceso que se está considerando. Un ejemplo puede darse en la acogida que tendrá un nuevo producto, expresado en previsión de ventas, a su salida al mercado. Suele ser bastante común su uso en contextos políticos o sociológicos. En este tipo de predicciones, los métodos estadísticos adquieren un papel secundario. Lo importante es contar con un grupo de personas especializada en la materia, con un conocimiento amplio del proceso a estudiar y predecir.

En los métodos cuantitativos, sin embargo, se parte de la idea de que se posee almacenada información sobre el pasado del proceso a estudiar. Este tipo de información aparece en forma de series temporales. La estadística cobra aquí un papel fundamental, ya que gracias a ésta se podrá extraer la información contenida en los datos, como se ha explicado anteriormente en el análisis de la serie temporal y sus componentes. A su vez, dentro de los métodos cuantitativos, podemos encontrar dos clases de predicciones: condicionales e incondicionales. Las predicciones condicionales se llevan a cabo mediante modelos causales, esto es, se predicen variables condicionadas por otra variable. Se hablará más adelante de este tipo de predicciones ya que también han sido objeto de esta tesis.

Por otro lado, las predicciones incondicionales se realizan mediante métodos autoprotectivos, en los cuales el modelo de predicción sólo incluye valores actuales, pasados y futuros de la propia serie en estudio. A su vez, dichos métodos pueden estar basados en dos enfoques, el determinista o clásico, y el estocástico o moderno. El enfoque determinista es más adecuado cuando se dispone de un número limitado de observaciones mientras que el enfoque estocástico es más adecuado cuando las series son de mayor tamaño. Dado el gran tamaño de las series temporales que se tratan en esta tesis doctoral, nos centraremos en este último enfoque.

El enfoque estocástico está basado precisamente en la metodología de Box y Jenkins, que consiste en aplicar operadores diferenciales repetidamente a los datos hasta que los valores se asemejen a un proceso estacionario [64]. Podría entonces usarse, de nuevo, la teoría de los procesos estacionarios para modelar, analizar y predecir la serie transformada, y en consecuencia, también la serie original.

Los modelos más conocidos de la metodología de Box y Jenkins son:

- AR(p): sus siglas se refieren a modelos autorregresivos de orden p. Describen una clase particular de proceso en el que las observaciones en un momento dado t son predecibles a partir de las p observaciones previas del proceso más un término de error. Toma la forma:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + ... + \phi_p X_{t-p} + \varepsilon_t \tag{2.13}$$

- MA(q): sus siglas se refieren a modelos de medias móviles de orden q. En este modelo el valor actual puede predecirse a partir de la componente aleatoria de este momento y en menor medida, de los impulsos aleatorios anteriores. O en otras palabras, las observaciones en un momento dado t son determinadas a partir del valor del error en el instante t y una combinación lineal de los errores pasados. Toma la forma:

$$X_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q} \tag{2.14}$$

- ARMA(p,q): sus siglas se refieren a modelos autorregresivos de medias móviles y es una extensión de los modelos AR(p) y MA(q), ya que incluyen tanto términos autorregresivos como de medias móviles. De esta forma, un proceso ARMA será estacionario si lo es su componente autorregresiva y será invertible si lo es su componente de medias móviles.

- ARIMA(p,d,q): sus siglas se refieren a modelos autorregresivos integrados de medias móviles de orden *p, d, q*. De esta forma este modelo permite describir una serie de observaciones después de que hayan sido diferenciadas d veces, con el fin de extraer las posibles fuentes de no estacionariedad, resultando en un proceso del tipo ARMA(p,q).

Con la aparición del machine learning, también surgieron nuevas técnicas para la predicción de series temporales, ejemplificadas en los algoritmos de redes neuronales artificiales [54], ya que estos algoritmos permitían explorar un mayor número de potenciales modelos además de proporcionar un método de predicción no lineal. Con la evolución de la inteligencia artificial de métodos basados en reglas a métodos basados en datos, el campo estaba ya preparado para aplicarse a las series temporales. Además, esto permitió que las series fueran almacenadas con órdenes de magnitud hasta entonces impensables, ya que las propias técnicas de machine learning requerían de conjuntos de datos más grandes.

Las redes neuronales, en concreto, se usan normalmente para el reconocimiento de patrones, donde un conjunto de características (como pudiera ser una imagen) son introducidas a la red, realizándose entonces la tarea de asignar dichas características de entrada a una o más clases.

Otro uso que suele darse a las redes neuronales es para regresión no lineal, donde la tarea principal consiste en encontrar una interpolación suave entre los puntos. En ambos casos, toda la información relevante se presenta de forma simultánea. Por contra, la predicción de series temporales requiere el procesamiento de patrones que evolucionan con el tiempo, ya que, como se ha explicado anteriormente, el valor adecuado a un punto en concreto de tiempo depende, no sólo del valor actual de la variable observada, sino también de su pasado.

El uso de redes neuronales además incluía una nueva terminología, diferente de la que se venía usando en otros métodos. Así, en lugar de hablar de "modelo", se tiene una "red". En lugar de parámetros, las redes tienen "pesos", y en lugar de "estimación de parámetros" se tiene "entrenamiento de la red". De esta forma, una red neuronal puede considerarse como una red de unidades, tipo neuronas, organizadas en capas. La capa más baja contiene un conjunto de unidades de entradas y la capa superior contiene un conjunto de unidades de salida. Dichas unidades en cada capa están conectadas con las de una capa superior.

De esta forma, en una red neuronal son varias las especificaciones que deben llevarse a cabo antes de ejecutarla:

- Su arquitectura de interconexión: se refiere al número de capas y neuronas en la red y en la forma de conectarlas.
- Su función de activación: relaciona el valor de la salida de un nodo con la entrada a otro nodo.
- Su función de coste: evalúa la salida de la red (como la función de errores cuadrados).
- Su algoritmo de entrenamiento: dicho algoritmo cambia los parámetros de interconexión o pesos para minimizar la función de coste. El algoritmo de backpropagation es uno de los más utilizados [41].

La predicción de series temporales siempre ha sido un campo atractivo para utilizar redes neuronales desde casi el comienzo de las mismas. Ya en 1964, en [42] se utilizaron redes lineales adaptativas de Widrow para predecir el tiempo meteorológico. Tras el desarrollo del algoritmo de backpropagation, otros estudios [44] entrenaron sus redes no lineales para emular la relación existente entre el siguiente punto de la serie a predecir y sus predecesores para una serie temporal generada por ordenador. En [92] los autores abordaron el problema de buscar redes de complejidad adecuada para predecir series temporales en casos reales.

La desventaja que presentan las redes neuronales es que no permiten entender fácilmente los datos porque no presenta un modelo de forma explícita. Lo que proporciona se podría considerar como una aproximación *black-box*

de predicción. Sin embargo, las redes neuronales consiguen buenos resultados donde un método que presente un modelo explícito fallaría. De hecho, las redes pueden adaptarse a irregularidades y características inusuales de las series temporales de estudio. Algunas investigaciones han comparado las técnicas de redes neuronales con métodos de predicción convencionales, como en [84] y [40], en las que las redes alcanzaron mejores resultados que los demás métodos con los que se comparan.

Actualmente, otros nuevos algoritmos de machine learning han sido aplicados con éxito a la predicción de series temporales. Es el caso de deep learning [46] y de random forest (RF) [14] como demuestran los trabajos realizados en [85] y en [29]. El resultado, tras utilizar dichos algoritmos, ha sido comparado con los resultados obtenidos por el algoritmo propuesto en este trabajo de investigación.

2.1.3. Series temporales multivariantes

En análisis univariante toda la atención se enfoca en una única serie temporal. Cuando se consideran varias series temporales juntas o series temporales múltiples, tendríamos un problema de series temporales multivariante. Este tipo de problemas o procesos surge cuando se estudian varias series temporales relacionadas simultáneamente a lo largo del tiempo en lugar de estudiar una única serie, como sería el caso de los problemas univariantes. Los procesos de series temporales multivariantes poseen gran interés en un gran número de campos:

- Ingeniería: estudio del comportamiento simultáneamente de la corriente y el voltaje, o de la presión, temperatura y volumen.
- Economía: variaciones de los tipos de interés, masa monetaria y tasa de desempleo.
- Empresarial: volumen de ventas, precio y gastos en publicidad.

El objetivo que se persigue analizando este tipo de problemas multivariante no es sólo describir las propiedades de cada serie sino que también se estudia las relaciones dinámicas entre variables y, a su vez, se mejora las predicciones que sobre las series temporales se realicen utilizando la información adicional que las otras series pueden aportar. El modelo, en este caso, buscaría recoger las interrelaciones entre las diferentes series. De esta forma se puede comprobar que en un modelo multivariante cada variable también depende del comportamiento de otra u otras variables.

Para estos casos, la mayoría de la teoría básica que se aplica a las series temporales univariantes se puede aplicar de forma natural a series temporales multivariantes. Uno de los más utilizados es una extensión del modelo AR, el modelo de vectores autorregresivos (VAR) [78]. Este tipo de modelos no pertenece exactamente a la familia de modelos estocásticos de series temporales

desarrollados por Box y Jenkins. Sin embargo, pueden considerarse como una generalización al campo multivariante de los modelos AR. En la definición de este modelo, se planteaba la importancia de las relaciones dinámicas en los fenómenos económicos y la escasa información que la teoría económica, en la que habitualmente se basaban los modelos estructurales, aportaba sobre estas relaciones dinámicas. Se desarrolló entonces un marco alternativo de elaboración y análisis con modelos econométricos multivariantes dinámicos que trataba de solventar los problemas que presentaban las aproximaciones estructurales clásicas. Estos primeros modelos VAR también pueden denominarse modelos VAR sin restricciones (*Unrestricted VAR*), a partir de los cuales se han ido planteando distintas variantes metodológicas con mejoras [64]:

- Modelos VAR estructurales (SVAR): para este tipo de modelos se evita la necesidad de imponer una ordenación a priori, proponiendo una transformación alternativa basada en incorporar una matriz con restricciones según la propia teoría económica.

- Modelos VAR parciales (PVAR): en estos modelos se realizan unas especificaciones más parsimoniosas y, a su vez, estadísticamente más correctas, en las que cada ecuación podía contener un número diferente de parámetros, debido a la sobreparametrización (y a la consiguiente falta de significatividad individual de los parámetros del modelo) que inicialmente no se analiza en los modelos VAR.

- Modelos VAR Bayesianos (BVAR): estos modelos incorporan la óptica bayesiana y su objetivo básico consiste en incluir en la estimación conocimientos previos acerca del comportamiento de los parámetros de forma que se aminore la sobreparametrización.

Otras implementaciones de la metodología clásica para problemas multivariantes de series temporales han incluido las medias móviles, dando lugar a los modelos VMA y VARMA [89].

Pero al extender los modelos univariantes a los modelos multivariantes también surgen nuevos problemas. El enfoque del modelo VAR, por ejemplo, criticaba la imposición de restricciones de identificación no avaladas por evidencias empíricas, cuestionando la elección de las variables endógenas (variables explicadas o dependientes) y exógenas (variables independientes), enfocándose al análisis de las interacciones dinámicas. Por ello, en este tipo de modelos no se impone ninguna restricción, siendo tratadas todas las variables como endógenas.

Había entonces que considerar otro tipo de problemas de series temporales multivariantes, aquellos en los que se dispone de una variable exógena o variable independiente y que puede influir sobre otra variable. Así podemos encontrar, por ejemplo, la regresión múltiple cuyo objetivo es analizar un

modelo que pretende explicar el comportamiento de una variable endógena utilizando la información que proporciona los valores tomados por un conjunto de variables exógenas. El modelo se puede escribir entonces de la forma siguiente:

$$Y_t = \beta_0 + \beta_1 X_{1t} + \beta_2 X_{2t} + ... + \beta_k X_{kt} + \varepsilon_t \tag{2.15}$$

Los parámetros β_1, β_2, ..., β_t denotan la magnitud del efecto que las variables exógenas X_{1t}, X_{2t}, ..., X_{kt} tienen sobre la variable endógena Y_t. El parámetro β_0 representa el término constante o independiente del modelo mientras que ε_t representa el término de error de dicho modelo.

Otras técnicas que también consideraban la influencia de variables exógenas en el modelado de series temporales multivariante son los modelos de vectores autoregresivos, que en la literatura se mencionan como VARX [89], utilizándose la X para referirse a las variables exógenas. La forma general de un modelo VARX sería la siguiente:

$$V_t = \phi_0 + \sum_{i=1}^{p} \phi_i V_{t-i} + \sum_{j=0}^{s} \beta_i X_{t-j} + a_t \tag{2.16}$$

donde V_t sería una serie temporal de k dimensiones, X_t una serie de m dimensiones de variables exógenas, a_t una secuencia de vectores aleatorios independientes e idénticamente distribuidos, ϕ_i la matriz de coeficientes habituales VAR y β_i la matriz de coeficientes $k \times m$.

2.1.4. Predicción de series temporales multivariante

Antes de intentar predecir los posibles valores futuros de una variable es necesario elegir el modelo adecuado que defina el comportamiento de dicha variable. En un problema de predicción de series temporales multivariante, en la explicación de la variable o variables objeto de estudio intervienen factores externos, de ahí que también se le denomine análisis causal [90]. El término causalidad pudiera parecer demasiado fuerte dadas las posibles connotaciones que se le puede otorgar para describir la forma en que variables externas o exógenas inciden en la explicación de una variable. Sin embargo, ha adquirido gran importancia especialmente en el análisis estadístico de variables relacionadas temporalmente. La frontera entre los modelos ARIMA univariantes y los multivariantes fueron propuestos por Box y Tiao [13]. Mediante esta técnica se puede analizar la influencia de factores externos sobre la dinámica de una serie temporal. Siendo más específicos, dentro del campo de de los modelos causales, un primer tipo de modelos serían los modelos de transferencia, también presentes en los trabajos de Box y Jenkins [11]. Estos modelos conservan en gran medida la filosofía de modelización ARIMA y permiten analizar la relación dinámica existente entre dos series.

Algunos de los modelos causales más utilizados provienen de modelos de regresión dinámicos como los de Koyck [43] o los de Almon [4]. Dichos mode-

los de regresión dinámicos son modelos de regresión que intentan hacer frente a los problemas específicos que se plantean al trabajar con series temporales.

Otros modelos causales que se pueden considerar para predicción son los modelos de cointegración y los modelos VAR anteriormente mencionados. Los primeros surgen ante los problemas de regresiones espurias con series temporales señalados por Granger y Newbold [34], siendo algunos ejemplos los trabajos [33] y [22]. La idea de equilibrio subyace a estos modelos. En cambio, los modelos VAR, son de naturaleza multiecuacional y nacieron como alternativa a los modelos multiecuacionales clásicos, que dependían de la imposición de restricciones carentes a menudo de justificación adecuada. Es por ello que, para este tipo de modelos, se consideren tanto el pasado de la propia variable a predecir como el resto de variables que pueden incidir sobre la primera. Así, un modelo de predicción VAR(p) se puede definir de forma muy parecida a un modelo AR(p), poniendo como ejemplo que contáramos con dos series temporales, X la cual queremos predecir e Y que tiene cierta influencia sobre la primera:

$$X_t = \phi_1 X_{t-1} + ... + \phi_p X_{t-p} + \delta_1 Y_{t-1} + ... + \delta_p Y_{t-p} + \varepsilon_t \tag{2.17}$$

Los métodos clásicos de predicción AR, ARMA y ARIMA, mencionados en la sección anterior, también presentan una modificación para incluir el análisis con variables exógenas, llamados ARX, ARMAX y ARIMAX [12]. Incluso, algunas implementaciones de estos métodos en determinados software, como los que presenta MATLAB, permite la predicción de series temporales multivariantes, no sólo ya de la variable objetivo sino de todas las implicadas. Dichos métodos han sido utilizados para compararlos con el algoritmo propuesto en este trabajo de tesis.

Al igual que sucedía con los problemas de series temporales univariante, los nuevos algoritmos de machine learning también podían adaptarse y, por tanto, aplicarse a problemas de series temporales multivariantes. Cabe destacar de nuevo el uso de redes neuronales, como los trabajos [17] y [69] así lo demuestran. Otro de los algoritmos que también han sido utilizados en este ámbito son los random forests. Un ejemplo de ello se puede encontrar en [97]. Tanto para redes neuronales como para random forests existen paquetes en R que se han utilizado en esta tesis para comparar resultados con el algoritmo desarrollado (los paquetes *neuralnet* [59] y *randomForestSRC* [60] respectivamente).

Uno de los objetivos que ha perseguido este trabajo de investigación ha sido precisamente desarrollar un algoritmo multivariante para no sólo predecir una variable objetivo sino todas al mismo tiempo, pudiendo utilizar información tanto de la variable objetivo como de las demás variables supuestamente relacionadas, esperando obtener predicciones más precisas.

Es decir, si, como teníamos expresado en la Ecuación (2.15), contamos con las variables Y, X_1, X_2, ..., X_k y deseamos predecir sus h próximos valores deberíamos obtener una matriz como la que se muestra a continuación:

$$\begin{bmatrix} Y_{(t+1)} & X_{1(t+1)} & X_{2(t+1)} & \cdots & X_{k(t+1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Y_{(t+h-1)} & X_{1(t+h-1)} & X_{2(t+h-1)} & \cdots & X_{k(t+h-1)} \\ Y_{(t+h)} & X_{1(t+h)} & X_{2(t+h)} & \cdots & X_{k(t+h)} \end{bmatrix}$$

Es necesario resaltar que, una vez definidos los métodos de predicción univariantes y los métodos de predicción multivariantes, para el análisis de unos datos en concreto, puede surgir la duda de qué línea sería la más conveniente a seguir. Cuando se dispone de un modelo teórico adecuado, de información estadística suficiente y de una previsión precisa de los factores externos o variables explicativas es preferible utilizar un modelo multivariante como instrumento para la previsión. Además, como suele ser el caso, la predicción de una variable no depende sólo de la propia variable de estudio sino de algunas otras que tengan influencia sobre ella. Pero en muchas ocasiones no se dan simultáneamente alguna de las circunstancias mencionadas anteriormente o no se dispone de más variables que influyan en la variable de estudio o, si se dispone de ellas, estas pueden no tener gran influencia sobre la variable de estudio, siendo necesario recurrir al análisis de carácter univariante como el trabajo de esta tesis demuestra.

2.2. Predicción de series temporales Big Data y los nuevos frameworks de procesamiento Big Data

2.2.1. La predicción de series temporales Big Data

Como se ha comentado en la introducción, almacenar datos ya no supone un problema debido a los últimos avances tecnológicos. Este hecho ha derivado en que en los últimos años sea tal el volumen de información disponible que se haya empezado a hablar de Big Data en todos los ámbitos de la vida diaria. El campo de las series temporales no ha sido ajeno a este crecimiento, siendo además uno de los que más ha contribuido a dicho aumento. Muchos investigadores han abordado en sus trabajos el problema de series temporales Big Data.

En [68] los autores proponen una nueva forma de buscar y minar en series temporales de alrededor de un "trillón" de datos (en palabras de los propios autores). Para ello utilizan DTW (*dynamic time warpping*) en lugar de la distancia euclídea en los algoritmos de búsqueda, obteniendo resultados en tiempos de ejecución bajos. Las pruebas realizadas fueron detección y clustering de secuencias genéticas.

En [75] se propone una nueva forma de indexar y minar series de datos de gran tamaño, cercanos al terabyte, mediante la combinación de búsqueda exacta y búsqueda aproximada en las subrutinas de algoritmos de minería de datos.

Si nos centramos en el campo de la predicción de series temporales Big Data, han sido varios las investigaciones que han abordado el asunto. Uno de los algoritmos de machine learning que ha sido aplicado con éxito ha sido deep learning. Así, podemos encontrar en [85], el desarrollo de un algoritmo basado en deep learning para predecir los datos de una serie temporal de electricidad de demanda eléctrica de más de diez años medida cada diez minutos. Los resultados alcanzaron un error considerablemente bajo y, gracias a la computación distribuida sobre la que se ejecutaba el algoritmo, en tiempos bastante reducidos. Dicho estudio fue revisado y mejorado en [86] por los mismos autores, y de nuevo, aplicando deep learning sobre el mismo conjunto de datos y llevando a cabo un estudio previo y extenso sobre los parámetros más óptimos a elegir, consiguieron mejorar los resultados de la predicción. Además, realizaron un estudio de escalabilidad, utilizando diferente número de hilos y multiplicando hasta por 64 el tamaño original de los datos para analizar los tiempos de computación, comprobando que el algoritmo ofrecía buenos resultados, en lo que a tiempos se refiere, con tamaños de datos considerable. En [87], los autores utilizan deep learning para predecir series temporales de energía solar, usando una metodología de múltiples pasos para descomponer el problema en subproblemas de menor tamaño permitiendo horizontes de predicción arbitrarios. Los resultados alcanzados fueron comparados con los resultados obtenidos por otros algoritmos, como redes neuronales y PSF [50], mejorando ligeramente a estos. Además, un estudio de escalabilidad entre los métodos fue llevado a cabo para comprobar su funcionamiento con series temporales Big Data, demostrando que el algoritmo de deep learning crece linealmente, mientras que las redes y el PSF aumentan exponencialmente, en lo que a tiempos se refiere.

Otro de los algoritmos que ha sido aplicado con éxito a la predicción de series temporales Big Data ha sido random forests. En [29] los autores proponen varios métodos escalables de predicción, sobre una serie temporal de energía eléctrica, basados en random forests, regresión lineal y dos tipos de árboles, concretamente un árbol de regresión [66] y Gradient-Boosted Trees (GBT) [52]. Las experimentaciones realizadas demostraron que random forests ofrecía mejor resultado que los demás métodos, siendo regresión lineal el que peores resultados ofrecía. El tamaño de los datos inicial fue multiplicado varias veces (hasta 32) y se llevó a cabo un estudio de escalabilidad, para comprobar la adecuación de los métodos para series temporales Big Data. Se demostró que los árboles de regresión eran los que mejores tiempos obtenían, mientras que la regresión lineal era la que peores tiempos alcanzaba. El anterior estudio fue ampliado en [30], donde los autores, como en

el caso de deep learning expuesto anteriormente, revisaban y mejoraban los métodos propuestos, realizando un estudio más pormenorizado del tamaño óptimo de la ventana para generar los modelos. Los resultados de predicción fueron mejorados ligeramente, y demostraron que, aunque random forests obtenía la mejor predicción, era preferible usar un único árbol de regresión, puesto que aunque el error fuera levemente superior, el tiempo de ejecución era mucho menor. Las pruebas de escalabilidad, estudiando número de hilos por ordenador y el tamaño del conjunto de datos, demostraron que de entre todos los métodos el árbol de regresión es el que ofrecía mejores tiempos de respuesta. Un paso más allá fue dado por los autores del estudio [28], donde se desarrolló un nuevo método, implementando un algoritmo que combina random forests, árboles de regresión y GBT. El algoritmo otorga pesos a cada método, siguiendo la técnica de mínimos cuadrados, según la precisión en su predicción. La forma de actualizar los pesos, dinámica o estáticamente, también se analiza en el estudio. Los resultados de la predicción del nuevo algoritmo, sobre dos series temporales de electricidad, mejoran a la predicción de cada uno de los métodos por sí solos. Los autores también comparan los resultados con redes neuronales, PSF y deep learning, a los que el algoritmo propuesto mejora.

Algunos algoritmos de clustering, incluso, han sido adaptados para predecir series temporales de tamaño que podría considerarse Big Data. Un ejemplo de ello lo podemos encontrar en [63], donde se desarrolla un algoritmo de k-means [49] para Big Data aplicado a una serie temporal de tres años de consumo eléctrico en tres edificios de una universidad obteniendo bajos errores en la predicción. El nuevo algoritmo es comparado, en términos de tiempos de ejecución, con Weka [94], conocido software para minería y análisis de datos, multiplicando el tamaño de los datos hasta por 300. Los resultados demuestran como el algoritmo incrementa los tiempos linealmente, de forma muy suave, a medida que crece el número de años a analizar indicando su predisposición para analizar Big Data, mientras que los tiempos con Weka crecen exponencialmente. Dicho trabajo fue ampliado más tarde en [62], para tratar de buscar patrones de consumo eléctrico en entornos de Smart Cities. Para ello los autores realizan un estudio previo seleccionando el mejor índice de validación de clustering para, posteriormente, aplicarlo al algoritmo en sí. Se utilizaron datos de consumo eléctrico de ocho edificios de una universidad pública tomados durante seis años para validar el algoritmo, resultando en el reconocimiento claro de ciertos patrones de consumo. Para simular un entorno Big Data y comprobar la validez del algoritmo, los investigadores crearon datos sintéticos de hasta 65536 edificios, lo que resultó en tamaños cercanos al terabyte. Las experimentaciones ofrecieron tiempos de ejecución inferiores a 4 horas para tal cantidad de datos.

En los últimos años ha ganado tanta popularidad entre la comunidad científica el estudio del futuro de este tipo de series temporales que algunos

autores han intentado resumir y recabar el mayor número de trabajos hasta el momento. En [38], más de 50 publicaciones de investigación fueron revisadas en el ámbito de la predicción en Big Data, haciendo especial hincapié en las series temporales. Entre las conclusiones obtenidas destacan:

- Los modelos factoriales son los más utilizados para predicción, seguidos de redes neuronales y modelos Bayesianos.
- El campo de la economía es el que cuenta con más publicaciones, esto es debido a que es del que más datos se encuentra disponible y por la propia importancia del campo en sí. El estudio del PIB y de la política monetaria son, dentro de la economía, los más estudiados, seguidos por los campos de dinámicas poblacionales y de la energía.
- Los mayores retos que, para el futuro, plantea el Big Data para obtener mejores predicciones, en palabras de los autores, son el desarrollo de habilidades relacionadas con el campo en los nuevos investigadores, el uso de nuevas técnicas de filtrado de ruido en las series o un mejor uso del hardware y software entre otros.

Dado el gran tamaño de las series temporales en estudio, algunos trabajos en el campo han empezado a focalizarse en la manera de reducir los tiempos de análisis y predicción, en lugar de buscar o desarrollar el algoritmo más ajustado y preciso para realizar la predicción. En [57] se analizaron 334 series temporales de varios tamaños, y se comparó la predicción que se realizaba sobre ellas con métodos de selección de párametros subóptimos (es decir, los que parecían no ser los más adecuados del todo) contra selección de parámetros más ajustados y exactos. El resultado fue que los subóptimos no ofrecían mucho peores resultados que el resto de métodos, y además permitían un decremento significativo y sustancial en los tiempos de computación de los mismos.

2.2.2. Los nuevos frameworks de procesamiento Big Data

Tradicionalmente, la computación a gran escala implicaba procesamientos complejos sobre pequeños conjuntos de datos. Herramientas muy populares y de gran potencia hasta entonces para el procesamiento y análisis de datos, como KEEL [2], R [67] o Weka, no fueron desarrollados para gestionar grandes cantidades de información. Así, con el crecimiento exponencial de los datos, se desarrolló una nueva forma de trabajar: la computación distribuida (Weka cuenta con implementaciones distribuidas para algunos de sus paquetes, pero no fue diseñada para funcionar de esa manera). Pero, como señalaba el estudio [38], uno de los mayores problemas a la hora de analizar Big Data reside en la configuración hardware y en los recursos software que se utilizan, debido a que se hace casi imposible escalar tal volumen de datos generados.

Aparecieron entonces nuevos frameworks de trabajo que permitían la implementación de algoritmos de data mining de forma distribuida para aprovechar al máximo las características de las máquinas en las que se lanzaba y a su vez ofrecían tiempos de respuesta aceptables. El más conocido es el paradigma MapReduce [20] desarrollado por Google en 2004. Se basaba en el principio de *divide y vencerás*, o dicho de otra forma, es un paradigma que permitía a los desarrolladores distribuir la computación de un algoritmo sobre un clúster de máquinas, dando como resultado un procesamiento de los datos más rápido y eficiente. Esto es debido a que dicha computación se realizaba sobre cada trozo de los datos al mismo tiempo en cada una de las máquinas del clúster en las que estuviera repartida la información. MapReduce permitía que la distribución de la información se llevara a cabo de manera automática, lo que lo dotaba a su vez de una gran escalabilidad ya que un programa podía funcionar sobre cualquier clúster de nodos, ya fuera con sólo 2 máquinas o con 1000.

Para poder procesar los datos, MapReduce primero requería que estos se estructuraran en tuplas del tipo <clave, valor>. Una vez se contaba con esa organización de los datos, se podían llevar a cabo las operaciones fundamentales que MapReduce soportaba y que le otorgan su nombre: *Map y Reduce*. La operación *Map* es la que procesa los bloques de información mientras que la operación *Reduce* fusiona los resultados de acuerdo a la clave que estos tengan. MapReduce soportaba así mismo dos operaciones más, que se consideran auxiliares: *Splitting y Shuffling*. El *Splitting* realiza la división de los datos en bloques mientras que el *Suffling* realiza las etapas de agrupamientos por la clave que posean los datos. Así, el flujo normal del funcionamiento de un programa en MapReduce con las operaciones anteriormente definidas se realizaría como se muestra en la Figura 2.2.

Figura 2.2: Etapas de ejecución de un programa en MapReduce.

Uno de los primeros software que permitían la implementación de MapReduce fue Apache Hadoop [31], cuya principal ventaja radicaba en su facilidad para implementar tareas analíticas sobre Big Data sin grandes conocimientos previos. Los usuarios podían codificar sus consultas usando Java en lugar de SQL, como se venía haciendo anteriormente en acceso a bases de datos. Esto suponía que no se requería formación previa, solamente un conocimiento básico en Java.

El framework Hadoop se basaba un sistema de ficheros distribuidos en los discos duros de los ordenadores de un clúster para la lectura y escritura de datos, llamado *Hadoop Distributed File System* (HDFS). Hadoop desarrolló un nuevo concepto de computación: distribuir los datos cuando estos

son almacenados para llevar a cabo la computación donde se encuentren los datos. Así, las principales características que Hadoop presentaba eran:

- Las aplicaciones en Hadoop eran implementadas en código de alto nivel.
- Los nodos pertenecientes al clúster se comunicaban entre ellos lo menos posible.
- Los datos eran distribuidos con antelación, lo que permitía que la computación se llevara a cabo donde se encontraban los datos.
- Los datos eran replicados para aumentar su disponibilidad y fiabilidad.
- Era escalable, lo que significaba que si se añadían más nodos al clúster aumentaba la capacidad del mismo proporcionalmente. Esto hacia que si, a su vez, se aumentaba la carga de trabajo el rendimiento disminuyera muy levemente sin llegar a producirse un fallo en el sistema.
- Tolerancia a fallos, de tal forma que, si un nodo fallaba, el sistema podía continuar trabajando, ya que las tareas se reasignaban a un nodo diferente y debido a la replicación de los datos no se producía perdida de estos. Además, si el nodo se recuperaba, se podía unir de nuevo al clúster de forma automática.

Todo ello conllevó que Hadoop alcanzara gran popularidad entre la comunidad científica. Algunos de dichos estudios, centrados en series temporales, son nombrados a continuación. En [71], se usó Hadoop para analizar una serie temporal consistente en mensajes de Twitter sobre acciones de empresas y evaluar su nivel de correlación con cambios en los precios de las acciones y en el volumen de negocio. En [9], se propone un nuevo framework de análisis de series temporales con datos de voltaje y extracción de características, basado en Hadoop, para intentar extraer dependencias en una red eléctrica. En [56], los autores desarrollaron una nueva librería para Hadoop, llamada Hadoop.TS, para el procesamiento de series temporales de gran tamaño. La librería proporciona además seis algoritmos de análisis de series temporales. Un nuevo método para descubrimiento de motivos en series temporales (subsecuencias similares o patrones frecuentes) de gran tamaño es propuesto en [48]. Los autores implementaron dicho método en Hadoop y fue probado con datos hospitalarios de electrocardiogramas. En [36] se propone una nueva forma de detectar anomalías en sistemas MapReduce combinando series temporales que contenían métricas de rendimiento de los sistemas operativos con información del sistema recogida en logs. Dichas anomalías son detectadas llevando a cabo clustering, mediante un algoritmo k-means, descubriendo patrones de contexto y patrones en las métricas. Toda la implementación y la experimentación fue realizada en Hadoop. Otro estudio sobre detección de anomalías fue realizado en [45]. Concretamente, los autores proponen un

nuevo framework, escalable y genérico, para descubrir anomalías automáticamente en series temporales de gran tamaño. Todo la infraestructura de análisis del framework fue desarrollada en Hadoop y permitía la detección de dichas anomalías en tiempo real. La experimentación fue realizada con series temporales reales de Yahoo (varios millones de series, en palabras de los autores).

Es importante reseñar que Hadoop mostraba ciertos problemas a la hora de desarrollar algoritmos iterativos, lo que hizo que surgieran otras propuestas en el mercado. Entre ellas, Apache Spark [98], desarrollado inicialmente en la universidad de Berkeley (California), es un software open source, que demostró ser una de las soluciones más potentes al llevarse a cabo toda la computación en memoria. Ello hacía que la ejecución de un algoritmo o programa fuera significativamente más rápido que en Hadoop (hasta 100 veces más rápido según la web oficial de Spark). De hecho, la velocidad de Spark fue puesta a prueba y comparada con otros métodos en 2014 en Daytona, en una competición que acabaría ganando [61]. El anterior récord era ostentado por Hadoop, el cual, para ordenar 102 TB repartidos en 2100 nodos tardó 72 minutos. Spark, sin embargo, ordenó 100 TB repartidos en 206 nodos en 23 minutos, es decir, tres veces más rápido con la décima parte de ordenadores disponibles. Como curiosidad, cabe mencionar que, un tiempo más tarde, ordenó un petabyte de datos repartidos en 190 nodos en 234 minutos [58].

El principal objetivo que perseguía Spark era generalizar las implementaciones MapReduce para soportar así nuevas aplicaciones dentro del mismo motor. Así, las principales características que ofrece este framework son:

- Capacidad para manipular datos en archivos batch o datos en HDFS, de forma interactiva pudiendo incluso llevar a cabo análisis en tiempo real.
- Posibilidad de desarrollar las aplicaciones en Java, Python o Scala (lenguaje de programación funcional que funciona sobre un JVM y en el que está escrito Spark).
- Permite programar a un nivel más alto de abstracción.
- Es de propósito más general, es decir, map/reduce es sólo uno de los conjuntos de constructores soportados.

El núcleo de Spark trabaja con variables propias, denominadas RDD. Un RDD representa un conjunto particionado de elementos que pueden ser operados en paralelo y cuyo valor no puede modificarse (son inmutables). RDD debe sus siglas a:

- *Resilient*. Se podría traducir como resistencia, ya que si los datos en memoria se pierden, pueden ser recreados de nuevo.

- *Distributed*. Distribuido y almacenado en memoria sobre el clúster de ordenadores.

- *Dataset*. Conjunto de los datos.

Así, cuando una serie de datos son leídos en Spark, automáticamente son cargados en un RDD y distribuidos en el clúster de ordenadores como se muestra en la Figura 2.3.

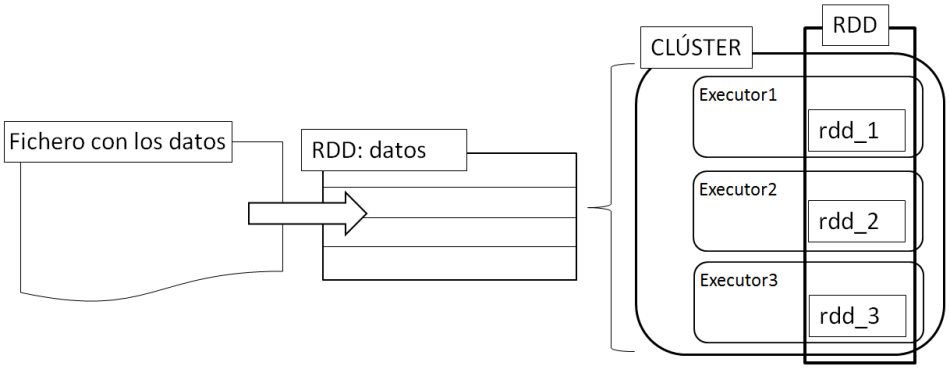


Figura 2.3: Creación de una variable RDD en Spark.

De esta forma, en Spark una serie temporal sería leída desde el origen de los datos y cargada en un RDD. Al distribuirse los datos por el clúster, es muy importante mantener el orden cronológico de los valores, siendo necesario otorgarle a cada valor un índice numérico dependiendo de su posición. Esto se consigue gracias a la función *zipWithIndex*, incluida dentro de la API de RDD. Esta asignación de índice se lleva a cabo una vez distribuidos los datos y sin pérdida de orden alguno, ya que Spark guarda un registro de la forma en que ha particionado los mismos, y siguiendo después el orden de los elementos en cada partición en sí.

La API de RDD fue extendida en 2015 para incluir DataFrames, que permiten a los usuarios agrupar una colección de datos distribuidos por columnas, de forma parecida a una tabla en una base de datos relacional. Pueden considerarse como RDD's con esquema.

Es importante resaltar que Spark no sólo permitía el desarrollo de nuevos algoritmos y funcionalidades, sino que también ofrecía algunos ya implementados en una librería del framework llamada MLlib [27]. Así, se puede llevar a cabo desde estadística básica sobre datos como cálculo de máximo, mínimo, varianza o correlación hasta estudios más intensivos de machine learning. Entre estos últimos destacan algoritmos de clasificación y regresión como naive Bayes o árboles de regresión y algoritmos de clustering como k-means. Spark cuenta así mismo con otra librería especializada, GraphX [26], pero en este caso para el análisis y estudio de grafos.

Una vez enumeradas las bondades que ofrece Spark, no sólo en tiempos de ejecución sino a la hora de implementar nuevos algoritmos o usar alguno de los ya existentes, cabe destacar como su uso se ha extendido, hasta casi ser un estándar de facto, entre la comunidad científica arrojando resultados muy satisfactorios sobre todo en el campo de las series temporales. Concretamente, en la predicción de series temporales de gran tamaño, ya han sido mencionados en anteriores secciones algunos de estos trabajos:

- En [29] y [30], se utilizaba el algoritmo de MLlib random forests.
- En [28], se utilizaba un algoritmo que combina random forests, un árbol de regresión y GBT, todos ellos incluidos en la librería MLlib.
- En [63] y [62] se hacía uso del algoritmo k-means de MLlib.
- En [85], [86] y [87] se llevaban a cabo diferentes experimentaciones sobre series temporales aplicando un algoritmo de deep learning en Spark.

Pero estos no son los únicos trabajos que han utilizado Spark con series temporales.

En [5], los autores aplican deep learning en análisis de grandes cantidades de datos de series temporales, extraídos de dispositivos móviles, tales como smartphones o IoT. Todo el desarrollo del algoritmo está implementado sobre Spark, donde los autores proponen que cada nodo desarrolle su propio modelo (con varias capas ocultas y millones de parámetros) y el ordenador principal o máster, construya el modelo final haciendo una media de los parámetros de los modelos parciales. El problema consistía en etiquetar una serie de muestras extraídas del acelerómetro de varios móviles. Las posibles etiquetas se correspondían con la posible actividad que se estuviera efectuando, que podía ser andar, correr, subir escaleras, estar de pie o estar tumbado. Para ello se contaba con unos 3 millones de muestras etiquetadas para construir el modelo de deep learning y aplicarlo a más de 38 millones de muestras sin etiquetar. Los resultados obtenidos fueron muy prometedores, teniendo la menor precisión en la clasificación entre estar tumbado y sentado. Una experimentación sobre la escalabilidad del algoritmo en Spark fue llevado a cabo, demostrando que la velocidad se reducía hasta 4 veces si se aumentaba el número de cores en el clúster de ordenadores.

En [77] Spark se utiliza como la herramienta de análisis, en tiempo real y off-line, de los datos recogidos por una red eléctrica inteligente (redes eléctricas que cuentan con varios dispositivos IoT que toman diferentes mediciones). En tiempo real, Spark es utilizado para monitorizar frecuencia, oscilaciones y voltaje además de para estimar el estado de la red. Los algoritmos de machine learning incluidos en la librería MLlib son aplicados a los datos, de forma off-line, para validar el modelo de los mismos y para llevar a cabo un análisis de alteraciones en la red.

Un uso de los árboles de regresión se puede encontrar en [72]. Concretamente los algoritmos disponibles en la librería de Spark MLlib, ejecutados sobre datos de series temporales de electricidad en Francia. El objetivo del método es construir un modelo que explique los patrones de consumo eléctrico dependiendo de diferentes características. Experimentaciones para probar la escalabilidad del método fueron, así mismo, llevadas a cabo, aumentando el tamaño de los datos hasta sobrepasar el terabyte de tamaño, demostrando que los tiempos subían linealmente y su adecuación al análisis de series Big Data.

Aunque Hadoop y Spark son de los frameworks más utilizados para el procesamiento de Big Data, es necesario mencionar otros competidores surgidos en años recientes:

- Apache Storm [47]. Framework para el procesamiento de datos en tiempo real, fue concebido para subsanar las deficiencias de otros procesadores al almacenar y analizar datos de redes sociales. Storm fue desarrollado en la compañía Backtype, empresa especializada en análisis de redes sociales, hasta que fue adquirida por Twitter en 2011, año en que se lanzó su primera versión. Storm es open source y desde que pasará a formar parte de Apache en 2014, se ha convertido en uno sus proyectos líderes. Debido a la importancia de los procesos en tiempo real, Storm ha ido ganando popularidad en los últimos años entre la comunidad de desarrolladores de machine learning. La estructura de Storm consiste en *spouts* y en *bolts*. Un *spout* se corresponde con el flujo de datos de entrada (como podría ser una API de Twitter) mientras que los *bolts* contienen la lógica de la computación. Storm está implementado en el lenguaje de programación Clojure, aunque también permite desarrollar en Java así como el uso de Thrift, un framework de desarrollo multi-lenguaje. Storm, al igual que Hadoop y Spark, posee una arquitectura que le permite tener tolerancia a fallos, pero no cuenta con librerías propias de machine learning, aunque es posible utilizar otras plataformas con integración con Storm, como SAMOA, que sí cuentan con técnicas de minería para flujo de datos de entrada en tiempo real Big Data.
- Flink [16]. Desarrollado en la universidad técnica de Berlín con el nombre de *Stratosphere* en el año 2014 [3]. En enero de 2015 pasó a ser un proyecto dependiente de Apache y hoy en día es considerado otro sus proyectos líderes. Flink puede procesar tanto ficheros de texto como flujo de datos de entrada en tiempo real. Los lenguajes de programación que soporta son Java, Scala, Python y SQL. El modelo de procesamiento de Flink realiza transformaciones sobre colecciones de datos paralelizados, generalizando las funciones *map* y *reduce*, así como las funciones de unión, agrupación e iterativas. Flink además cuenta

con un optimizador, basado en coste computacional, lo que hace que automáticamente seleccione la mejor estrategia para cada ejecución o trabajo. Flink ML es una librería de machine learning disponible a partir de abril de 2015, aunque Flink también cuenta con adaptaciones disponibles para la librería SAMOA anteriormente comentada.

- H_2O [19]. Framework open source que proporciona un motor de procesamiento paralelo, librerías analíticas, matemáticas y de machine learning, así como herramientas de preprocesamiento y evaluación. También ofrece una interfaz de usuario web, lo que hace que programar tareas en H_2O esté al alcance de casi cualquier persona dado que no requiere grandes conocimientos de programación. Aún así, ofrece soporte para Java, R, Python y Scala para programadores más avanzados. El motor de H_2O procesa los datos en memoria usando múltiples métodos de ejecución, dependiendo del que sea más eficiente para cada algoritmo utilizado. El planteamiento general de H_2O usa un *Fork/-Join* distribuido, lo que no es más que una técnica de divide y vencerás, la cual es adecuada y segura para tareas masivas en paralelo. Este método permite separar una tarea en subtareas más pequeñas trabajando en paralelo, dando como resultado un balanceo de carga dinámico para las ejecuciones MapReduce así como para grafos y flujo de datos en tiempo real. H_2O, a partir de su versión 3, también ofrecía lo que se denominó como *Sparkling Water*, que no es más que una plataforma que permitía la integración de H_2O con Spark y Spark Streaming.

Capítulo 3

Detalles metodológicos

En la antigüedad se usaban bueyes para realizar trabajos pesados. Pero cuando un buey no podía mover un tronco, no se intentó criar un buey más grande. De igual forma, no deberíamos trabajar con ordenadores más "grandes", sino con clústers de ordenadores.

Grace Murray Hopper (militar estadounidense y primera persona que desarrolló un compilador)

Resumen: En este capítulo se detalla la metodología seguida a la hora de desarrollar el algoritmo de vecinos cercanos para predicción de series temporales Big Data, tanto en su versión univariante como en su versión multivariante, todo ello bajo el framework de Apache Spark. Lo primero que se aborda es cómo se comenzó a implementar el algoritmo propuesto para adaptarse a dicho tipo de series temporales de gran tamaño. Seguidamente, se comentan las mejoras introducidas, entre ellas la selección automática de los parámetros óptimos para el algoritmo y se finaliza explicando la adaptación del mismo para entornos multivariantes.

3.1. Implementación y desarrollo de un algoritmo de predicción de series temporales Big Data

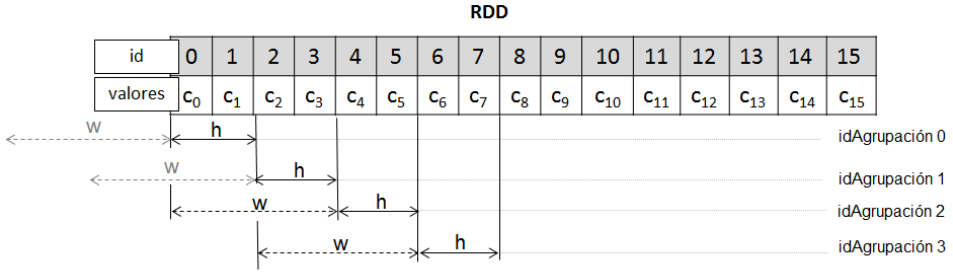
El objetivo que persigue esta tesis es desarrollar un algoritmo de predicción de series temporales en entornos Big Data para poder aplicarlo en el contexto de Smart Cities, como se comentaba en la Sección 1.2 del Capítulo 1. Más concretamente, el algoritmo de vecinos cercanos o kNN. Pero

la complejidad de este algoritmo es del orden $O(n \cdot D)$ (siendo n el número de instancias y D el número de características) sólo para encontrar el vecino más cercano de una determinada instancia. Por lo cual, encontrar los k vecinos más cercanos añade un nivel de complejidad mayor, dado que las distancias calculadas necesitan ser ordenadas, resultando en una complejidad de orden $O(n \cdot log(n))$. Debido a este elevado coste y trabajando a su vez con grandes cantidades de datos, los framework de programación clásicos no pueden dar una respuesta adecuada en lo que respecta a tiempos de computación. Las ventajas que presenta Apache Spark pueden ofrecer una solución para la implementación del algoritmo, al ofrecer una computación distribuida. Además, dicho algoritmo utiliza el lenguaje de programación Scala, en el que está desarrollado el propio framework de Spark, y que ofrece una visión radicalmente diferente a los lenguajes de programación tradicionales, como Java o C, al tratarse de un lenguaje de programación funcional y por tanto, idóneo para la computación distribuida.

De tal forma, se comienza adaptando el algoritmo kNN a dicho lenguaje para que, además, se ejecute sobre un clúster de ordenadores con Spark. Es necesario abstraerse del resultado final, el cálculo de las distancias y la selección de los vecinos más cercanos, y centrarse en una implementación que se lleve a cabo en cada trozo del conjunto de datos original repartido en los ordenadores del clúster. Así, es de vital importancia mantener el orden cronológico de los datos, objetivo conseguido utilizando la función *zipWithIndex*, como se ha explicado en el Apartado 2.2.2 del Capítulo 2. Una vez llegado a este punto, el algoritmo kNN requiere agrupar los valores en igual número que el patrón a clasificar para poder compararlos (w valores) y así después calcular las distancias. Pero el algoritmo desarrollado en esta tesis, el kWNN (*k weighted nearest neighbours*) es una variación del original y está inspirado en el weighted nearest neighbours (WNN) [88] usado no para clasificar, sino para predecir. De esta forma, se requiere una variable más, h que representa el horizonte de predicción o número de valores a predecir. El funcionamiento del algoritmo consiste en que, dado un vector de w valores, para predecir sus h siguientes, se busca los k vecinos más cercanos a dicho vector. Después, se calcula un peso para cada vecino en función de la distancia al vector y se realiza la predicción aplicando una media de los h siguientes valores a cada vecino ponderada por dichos pesos. De tal forma que el algoritmo necesita formar todas las posibles ventanas de w valores cuyos h siguientes sean conocidos, dado que dichas ventanas de h valores de los k vecinos más cercanos son usadas para predecir.

La creación de vectores de h valores a partir de la serie temporal original no supone un problema, ya que todas las ventanas de datos son consecutivas y sin repetición, como se muestra en la Figura 3.1, donde h representa 2 valores. Así, obteniendo el cociente entero de la fórmula id/h se consigue el *idAgrupación* en el que se encuadra dicho dato en las ventanas de h.

En la Figura 3.1, puede comprobarse como c_0 y c_1 se agrupan, siguiendo dicha fórmula, en la ventana con *idAgrupación* 0 mientras que c_6 y c_7 se agrupan en la ventana con *idAgrupación* 3.



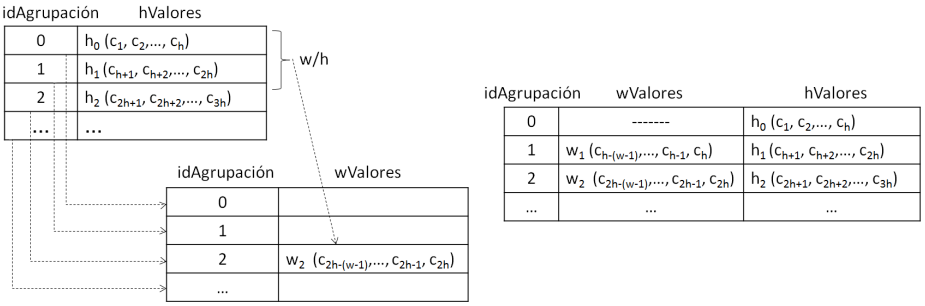
idAgrupación	wValores	hValores
0	----	(c_0, c_1)
1	(--,--, c_0, c_1)	(c_2, c_3)
2	(c_0, c_1, c_2, c_3)	(c_4, c_5)
3	(c_2, c_3, c_4, c_5)	(c_6, c_7)
...	...	...

Figura 3.1: Formación de ventanas de w valores y sus siguientes h valores en Spark.

Pero la creación de vectores con los w valores anteriores a cada una de las ventanas h supone un paso más complejo en el desarrollo del algoritmo, ya que es complicado encontrar la forma en que, de forma simultánea sobre cada trozo de la serie temporal, se calcule a qué ventana w de valores pasados pertenece cada dato, pudiendo éste estar repetido en varias ventanas como se muestra en la Figura 3.1. Se plantea en primera instancia una solución iterativa y dependiente de la formación de las ventanas de h valores. De tal forma que, para obtener los vectores de w elementos pasados, se toman los valores de las w/h ventanas formadas por elementos anteriores a los h valores a predecir, como se muestra en la Figura 3.2.

Una vez obtenidas las ventanas de w y h valores, se calcula la distancia de cada posible vecino al patrón de referencia que se esté analizando, se ordenan los vecinos y se seleccionan los k más cercanos, llevándose a cabo la predicción aplicando una ponderación de pesos sobre las ventanas de h valores futuros seleccionadas, según la fórmula que se encuentra en el trabajo publicado en [83]. Dicha ponderación otorga más incidencia en la futura predicción a los vecinos que de verdad sean más cercanos, de manera que penaliza a los más alejados, aunque sigan influyendo en dicha predicción.

De esta forma, el algoritmo comienza dividiendo el conjunto de datos de entrada en un 70 %, con el que se realiza el entrenamiento (TRS), y el restante 30 % se usa para predecir de h en h valores el conjunto de test (TES), y comprobar así su validez calculando el error relativo medio (MRE).



(a) Datos a partir de las w/h ventanas anteriores.

(b) Ventanas de w y h completas.

Figura 3.2: Forma iterativa de obtener las ventanas de w valores pasados a los h valores a predecir.

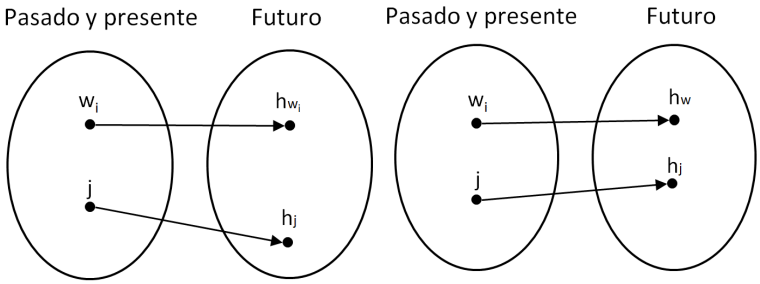
Una explicación más detallada del funcionamiento del algoritmo así como la ecuación del MRE pueden encontrarse en el artículo [83].

3.2. Mejoras y adaptación del algoritmo kWNN para Big Data

Que el algoritmo tenga la fase iterativa anteriormente mencionada hace, además, que los tiempos de ejecución del mismo aumenten considerablemente a medida que crece el tamaño de la serie temporal. Por lo que se ha buscado una solución distribuida, basada en una fórmula que se puede aplicar sobre cada trozo de los datos de entrada, particionados estos a lo largo del clúster. Así, para calcular el *idAgrupación* de ventanas de w en las que está presente cada dato, se usa el rango obtenido como sigue: $(id/h)+1$ hasta $(id/h)+(w/h)$. Un ejemplo, puede verse en la Figura 3.1. Siguiendo la fórmula, c_2 estaría en las ventanas de w_2 y w_3. Una vez solucionado este primer obstáculo, el algoritmo queda implementado de forma totalmente distribuida, lo que hace que bajen los tiempos de computación y permita probarse con conjuntos de datos de mayor tamaño.

Lo siguiente es valorar la idoneidad del peso elegido, al jugar éste un papel clave a la hora de realizar las predicciones, primando dar la mayor importancia e influencia posible a los vecinos más cercanos. Por ello, un total de 4 pesos distintos son estudiados, como se puede ver en [81]. Los resultados demuestran que el peso elegido en primer término, esto es $1/(dist)^2$, es el ideal, al ser éste el que mayor importancia otorga a los vecinos que más cerca se encuentran, por encima de los otros pesos.

Otra mejora se ha añadido al algoritmo, con la selección del tamaño óptimo de las ventanas w para obtener una predicción más precisa. Para ello se usa el método de los *falsos vecinos cercanos* [88]. Dicho método compara la distancia entre una ventana de w valores y su vecino más cercano (d_w) con la distancia entre los h valores siguientes a la ventana y los h valores siguientes al vecino más cercano (d_h). Si la segunda distancia es mayor que la primera ($d_h > d_w$), se considera que el vecino cercano encontrado era un falso vecino, porque la trayectoria de los h valores siguientes tiende a divergir, como se puede ver en la Figura 3.3 (a), donde w_i representa una ventana de w valores y j su vecino más cercano, siendo h_{w_i} los h valores siguientes a w_i y h_j los h siguientes valores a j. Se puede comprobar en este caso que la distancia entre w_i y j es menor que la distancia entre h_{w_i} y h_j. Sin embargo, en la Figura 3.3 (b) ocurre lo contrario, y como se puede apreciar, se obtendrían mejores predicciones en este escenario.



(a) Falso vecino cercano. (b) Vecino cercano óptimo.

Figura 3.3: Identificación de falsos vecinos.

De esta forma, este método intenta encontrar el tamaño de ventana w que minimice el número de falsos vecinos. Para ello se comprueban diferentes tamaños de w comenzando por $w = h$. Para llevar a cabo este método, el algoritmo kWNN divide el conjunto de datos de entrada como venía haciendo, es decir, un 70 % para TRS y un 30 % para TES. Después, se aplica sobre el TRS el método de los falsos vecinos, dividiendo a su vez el conjunto de entrenamiento en un 70 % (STRS) y un 30 % (STES).

El primer tamaño de w que baje de un 10 % el porcentaje de vecinos que son falsos se toma como el óptimo. Para más detalles sobre como funciona este método, consultar [81].

Otras de las mejoras agregadas al algoritmo es hallar el número óptimo de k vecinos cercanos a seleccionar. Este método en particular consiste en realizar la predicción sobre solamente el STES, una vez definido el tamaño óptimo de w y con diferentes números de vecinos. El que resulte en un menor MRE será el elegido por el algoritmo. Puede encontrarse la especificación del método en [81].

Diferentes tamaños de horizontes son analizados. En concreto se toman como horizontes 4 horas, 8 horas, 12 horas y 24 horas, y, para cada uno de ellos, se calcula la ventana w óptima así como el número k óptimo.

Una vez añadidos los anteriores métodos y mejorada la implementación del algoritmo, el kWNN está preparado para, automáticamente, seleccionar los parámetros óptimos necesarios y realizar la predicción de cualquier conjunto de datos que se le proporcione.

3.3. Versión multivariante del algoritmo: MV-kWNN

Suele ser algo bastante común en el mundo real, y sobre todo en problemas de series temporales, encontrarse con un conjunto de datos para el cual la predicción de su futuro no dependa tanto del comportamiento de éste en el pasado, sino de otras variables externas o exógenas al mismo, como se ha comentado en el Apartado 2.1.3 del Capítulo 2.

Por este motivo, se adaptó el algoritmo kWNN para que pudiera analizar no una serie temporal sino varias y, además, no sólo predecir una variable objetivo, sino todas las analizadas al mismo tiempo y teniendo en cuenta el comportamiento e interrelaciones entre ellas.

Se implementó de esta forma el MV-kWNN, la versión multivariante del algoritmo. El MV-kWNN está adaptado para tomar un número variable de series temporales, pudiendo además tener éstas diferentes intervalos de tiempo. Así, el MV-kWNN comienza agrupando cada serie temporal en ventanas de w valores y en ventanas de sus h siguientes valores, como hacía el kWNN. Pero w y h se expresan en esta versión de forma genérica en horas, en lugar de en número de valores. Después, el propio algoritmo, automáticamente, se encarga de agrupar de forma correcta cada serie temporal, dependiendo del número de valores por hora de la serie temporal. A continuación, el MV-kWNN une dichas agrupaciones para trabajar con un único conjunto de datos y respetando el orden de los mismos. Esto resulta en una única matriz con las ventanas de w valores de cada serie en primer término y las ventanas de h valores siguientes de cada serie temporal en segundo término. El algoritmo entonces actúa de igual forma que anteriormente, dividiendo el conjunto en TRS y TES, buscando los k vecinos más cercanos y realizando la predicción de todas las series temporales. De este modo, se buscan patrones de comportamiento parecidos en un instante de tiempo determinado de todas las variables en consideración a la vez. Esto puede apreciarse con más detalle en [82], donde se muestra además un ejemplo numérico a modo de apéndice.

Capítulo 4

Discusión de resultados

El mundo exige resultados. No les cuentes a otros tus dolores del parto, muéstrales al niño.

Indira Gandhi.

Resumen: En este capítulo se detallan los resultados obtenidos en los diferentes trabajos de investigación que componen esta tesis. Se comienza comentando los primeros resultados conseguidos tras aplicar el algoritmo a los datos de consumo eléctrico de dos edificios de una universidad pública medidos durante 3 años. Después, se detallan las nuevas experimentaciones realizadas tras mejorar el algoritmo univariante, utilizando esta vez como conjunto de datos de entrada la demanda eléctrica en España durante más de 9 años. Finalmente, y tras desarrollar la versión multivariante del algoritmo, se analizan los resultados obtenidos sobre la demanda y el precio de la electricidad en España, medidos durante más de 9 años.

4.1. Consumo energético en edificios de una universidad pública

Para llevar a cabo la primera experimentación del algoritmo desarrollado, se utilizan los datos de consumo eléctrico recogidos cada 15 minutos en varios edificios de la Universidad Pablo de Olavide (UPO) durante los años 2011, 2012 y 2013 como se muestra en la Figura 4.1. La primera columna se refiere al edificio, la segunda al consumo, y las siguientes se corresponden con el año, mes, día, hora y minutos respectivamente. En concreto, los edificios estudiados corresponden a la cafetería de la universidad y a un edificio donde se imparte docencia.

Destacar así mismo que, antes de utilizar el conjunto de datos, hubo de realizarse una tarea de limpieza previa de los mismos, ya que se encontró un considerable número de registros ausentes y outliers. De tal forma que, la siguiente medida que aparecía registraba el valor acumulado de los registros ausentes, teniendo entonces que crear dichos registros y dividir la cantidad acumulada entre el número de estos.

```
ELEC_EDIF_1;1.511;2012;01;01;00;00
ELEC_EDIF_1;1.549;2012;01;01;00;15
ELEC_EDIF_1;1.454;2012;01;01;00;30
ELEC_EDIF_1;1.475;2012;01;01;00;45
ELEC_EDIF_1;1.423;2012;01;01;01;00
ELEC_EDIF_1;1.391;2012;01;01;01;15
ELEC_EDIF_1;1.317;2012;01;01;01;30
ELEC_EDIF_1;1.838;2012;01;01;01;45
```

Figura 4.1: Conjunto de datos de consumo eléctrico en edificios de la UPO.

Para el tamaño de las variables necesarias para la correcta ejecución del algoritmo se eligen los siguientes valores:

- Se establece w a un día completo (lo que se corresponde con 96 registros).
- Se analizan dos horizontes de predicción: medio día (48 registros) y un día completo (96 registros).
- Para k se estudian varias posibilidades, desde 1 a 5 vecinos.

Los resultados obtenidos pueden encontrarse en la Tabla 1 y en la Tabla 2 del artículo [83]. Como puede observarse, se miden los tiempos de ejecución en minutos y la precisión de los resultados con el MRE, combinando los diferentes tamaños de h con w y con el número de k, para datos de 2 edificios, dando lugar a 4 experimentaciones. La primera conclusión extraída es que, como se esperaba, los tiempos de ejecución son casi idénticos para la experimentación con los datos de cada edificio al contar con el mismo número de registros. Sin embargo, los tiempos disminuyen entre las experimentaciones cuando se consideran horizontes de predicción mayores. Es decir, habiéndose establecido w en un día completo, disminuyen los tiempos en más de la mitad cuando h se establece, así mismo, a un día completo. Esto es debido al menor número de instancias totales para las que hay que calcular la distancia con el objeto de obtener los vecinos cercanos, al ser h de mayor tamaño.

Si nos centramos en los resultados obtenidos por cada edificio según los diferentes tamaños de h, se puede observar que el mejor MRE se obtiene con h establecido a medio día para ambos edificios:

- En el primer edificio, el mejor MRE supone un 44 % más bajo proporcionalmente que el mejor resultado para horizontes de predicción de 1 día.

- En el segundo edificio, el mejor MRE supone un 51 % más bajo proporcionalmente que el mejor resultado para horizontes de predicción de 1 día.

Por otro lado, en lo que respecta al MRE obtenido para cada número de vecinos cercanos diferentes, los resultados en las experimentaciones para ambos edificios son muy similares. En las 4 experimentaciones el menor MRE se obtiene con 2 vecinos cercanos. El mayor MRE, sin embargo, varía:

- Para el primer edificio:
  - Cuando w es establecido a un día y h a medio día, con 1 vecino cercano se obtiene el mayor MRE, aunque solamente un 2 % más alto que la mejor predicción.
  - Cuando w es establecido a un día y h también a un día, con 4 vecinos cercanos se obtiene el mayor MRE, en este caso un 5 % más alto que la mejor predicción.
- Para el segundo edificio:
  - Cuando w es establecido a un día y h a medio día, con 1 vecino cercano (igual que en el primer edificio) se obtiene el mayor MRE, aunque solamente un 3 % más alto que la mejor predicción.
  - Cuando w es establecido a un día y h también a un día, con 1 vecino cercano se obtiene el mayor MRE, en este caso un 7 % más alto que la mejor predicción.

Las conclusiones que se obtienen tras los resultados obtenidos son que un menor horizonte de predicción, h, supone una sustancial mejora en lo que respecta al MRE. Es decir, que el algoritmo parece ajustarse más a predicciones a corto plazo que a predicciones a más largo plazo. Por otro lado, no se estudian diferentes tamaños de las ventanas w, lo que unido a una adecuada selección del número óptimo de vecinos podría disminuir y mejorar la precisión de las predicciones, ya que hasta ahora dichos valores se tenían que establecer antes de que comenzara el algoritmo. Aunque, por otro lado, la poca diferencia entre los mejores y peores resultados para los diferentes números de vecinos cercanos evaluados (entre un 2 % y un 7 %) reflejan lo adecuado y robusto que demuestra ser el algoritmo. En lo que respecta a los tiempos de ejecución, se observa que estos aumentaban a más del doble dependiendo del tamaño de h, y por tanto, del número de predicciones a realizar. Esto también está influenciado por la propia implementación del algoritmo, al contar éste con una parte todavía iterativa, ya que para formar las ventanas de w se requiere consultar las ventanas formadas por valores anteriores a los h valores a predecir, tal como se describe en el Capítulo 3.

4.2. Demanda de energía eléctrica en España

4.2.1. Resultados con algoritmo univariante

Una vez mejorado el algoritmo inicialmente desarrollado, como se comenta en la Sección 3.2 del Capítulo 3, se analiza la precisión en las predicciones del mismo cuando se aplica a un conjunto de datos reales.

Para las experimentaciones se utiliza esta vez una serie temporal de mayor tamaño, dado que el algoritmo ya está adaptado para su ejecución de forma distribuida. Así, se usa la demanda de energía eléctrica de toda España, medida cada 10 minutos desde enero del año 2007 a junio de 2016, lo que supone 3 veces más datos que el anterior dataset formado con la demanda de energía eléctrica en dos edificios de la UPO. Un extracto de este conjunto de datos se puede ver en la Figura 4.2, donde la primera columna representa la fecha y el segundo campo la demanda eléctrica en sí (el resto de campos no se tienen en cuenta).

```
2007-01-01 00:00,28667,6626,1796,3775,5340,4540,4391,2640,-1179,0,0
2007-01-01 00:10,28204,6628,2148,3974,5617,3678,4499,2691,-761,0,0
2007-01-01 00:20,27781,6632,2273,4137,5520,3368,4526,2691,-544,0,0
2007-01-01 00:30,27139,6634,2319,4043,5429,3012,4584,2691,-464,0,0
2007-01-01 00:40,26687,6629,2180,3951,5082,2722,4801,2691,-518,0,0
2007-01-01 00:50,26333,6627,2168,3911,4846,2739,4867,2691,-829,0,0
2007-01-01 01:00,27611,6226,1899,4345,4377,4360,3784,2525,-1912,0,0
```

Figura 4.2: Conjunto de datos de demanda de energía eléctrica en España.

Se mencionará primero el análisis realizado y los resultados obtenidos para seleccionar el valor óptimo de w para los horizontes estudiados. Excepto cuando h representa 12 horas en el que el tamaño óptimo de w es 8 veces el tamaño de h, para los demás tamaños de h considerados, la primera vez que se obtiene menos de un 10 % de falsos vecinos se produce cuando el tamaño de w es 6 veces el tamaño de h. Se puede concluir que, para este conjunto de datos, el algoritmo necesita que el tamaño de las ventanas w sea mucho mayor (entre 6 y 8 veces más grande) que el número de valores a predecir para ser así más preciso. Esto puede apreciarse más en detalle en la Figura 4.3, donde los resultados obtenidos se pueden resumir de la siguiente manera:

- Caso 1: cuando h representa 4 horas w debe ser 1 día completo.
- Caso 2: cuando h representa 8 horas w debe ser 2 días completos.
- Caso 3: cuando h representa 12 horas w debe ser 4 días completos.
- Caso 4: cuando h representa 24 horas w debe ser 6 días completos.

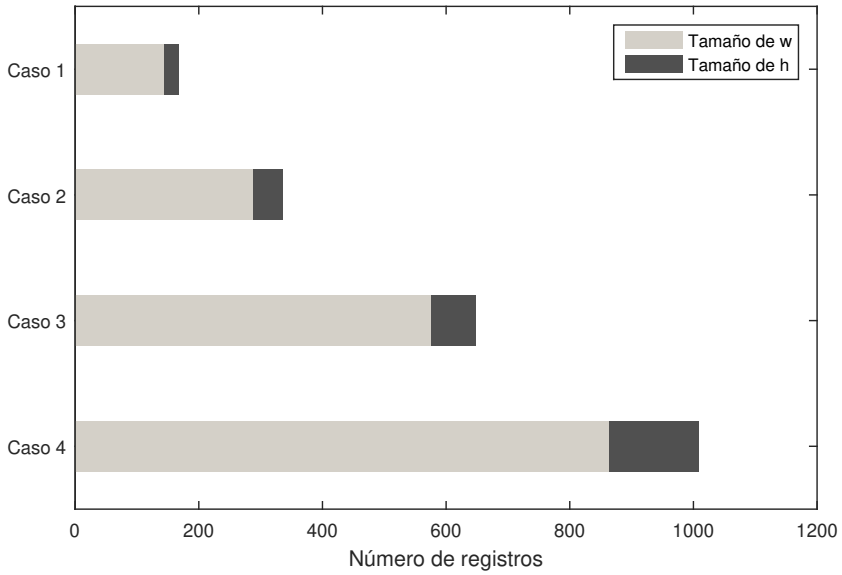


Figura 4.3: Tamaños óptimos de ventanas de w valores para diferentes horizontes de predicción.

Cabe destacar también que se valoraron tamaños de w más allá del primero que obtuviera menos de un 10 % de falsos vecinos, y como se puede apreciar en [81], el porcentaje de falsos vecinos continuaba descendiendo muy levemente, haciendo que el tamaño elegido fuera el ideal, ya que tamaños de w mayores no ofrecían sustanciales mejoras y podrían redundar en una mayor carga computacional del algoritmo.

Una vez seleccionado el tamaño de w, el cálculo del número óptimo de vecinos cercanos, k, es llevado a cabo como sigue: se valoran desde 1 hasta 20 posibles vecinos cercanos y se realiza la predicción sobre el conjunto de validación STES. Los resultados reflejan que, para todos los casos el menor MRE alcanzado se obtiene con 4 vecinos cercanos, excepto para el caso 2, cuyo número óptimo de vecinos resulta ser 2. En dichos resultados puede apreciarse como, además, el error se mantiene más o menos constante sin grandes variaciones en todos los casos para los diferentes números de vecinos, lo que demuestra que, aun no escogiéndose el número óptimo de vecinos en el algoritmo, se obtendría una predicción aproximada. Es importante destacar que el menor MRE se consigue en el caso 1, con un 1.90 %, es decir con un horizonte de predicción de 4 horas y un tamaño de ventana w de 1 día, como se puede ver en la Figura 4.4. En dicha figura se puede apreciar como a medida que crecen los tamaños de los horizontes de predicción los errores tienden a crecer aunque de forma muy moderada y linealmente hasta alcanzar un máximo de 3.36 %.

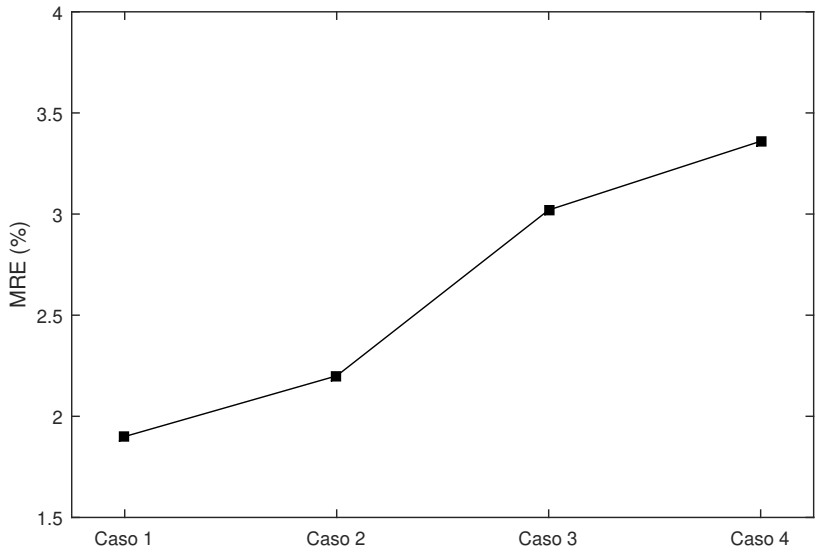


Figura 4.4: MRE obtenido con tamaño óptimo de ventana y vecinos cercanos para diferentes horizontes de predicción.

Para analizar en detalle las predicciones obtenidas con el algoritmo de predicción propuesto, nos centraremos en una ventana de tamaño w de un día, un horizonte de predicción h de 4 horas y selección de 4 vecinos cercanos (k) para realizar dicha predicción.

Así, se obtiene un 1.63 % de MRE en la predicción final sobre el conjunto de test de la demanda eléctrica en España, lo que demuestra no sólo la robustez e idoneidad del algoritmo, sino también la importancia de seleccionar los parámetros óptimos para el mismo. Es necesario reseñar en este punto que se predicen un total de 2 años y 10 meses, tamaño que representa el 30 % de los datos. En concreto, desde los últimos días de agosto del año 2013 hasta junio de 2016.

Analizando más en detalle la predicción realizada, se analizan los meses que mejor y peor MRE obtienen, como se puede ver en la Tabla 4.1. Los meses que peor predicción alcanzan en el algoritmo son siempre abril y diciembre, teniendo abril incluso el mismo error para el año 2014 que para el año 2016. Por otro lado, los meses que mejor predicción muestran son febrero junto con septiembre y octubre, alcanzando de nuevo el mismo porcentaje de error en los meses de febrero tanto en el año 2014 como en el año 2016.

Cabe destacar por otro lado que, sin embargo, los meses que en principio podrían considerarse más difíciles de predecir, como son los que comprenden el periodo estival al coincidir con el común periodo vacacional, se mantienen ligeramente por encima o incluso por debajo del error medio de la predicción,

pero sin llegar a ser los que mayor error arrojen, lo que demuestra que el algoritmo es capaz de adaptarse a los comportamientos cambiantes de dichos meses y predecir con bastante precisión la demanda asociada a los mismos.

Tabla 4.1: MRE de las mejores y peores predicciones de cada año.

Año	2013		2014		2015		2016	
Mejor mes	Sep.	1.55 %	Feb.	1.32 %	Oct.	1.36 %	Feb.	1.32 %
Peor mes	Dic.	2.03 %	Abr.	2.12 %	Dic.	1.81 %	Abr.	2.12 %

Tanto el mes con mejor predicción de entre todos como el que peor error obtuvo, se analizan con más detalle a continuación.

Febrero del año 2014, con un 1.32 % es el elegido como mejor predicción. Se observa que sólo 10 días del mes tienen un error superior al error medio del mismo, obteniendo el peor día un 2.47 % mientras que el resto de días se mantienen por debajo de dicha media, incluso alcanzándose en el mejor día un error inferior al 1 % (0.65 % para ser más exacto). En la Figura 4.5 se muestra la demanda real diaria para dicho mes frente a la predicción, donde se observa como ambas curvas presentan una gran similitud solamente diferenciadas por los 10 días de error superior a la media del mes mencionados anteriormente.

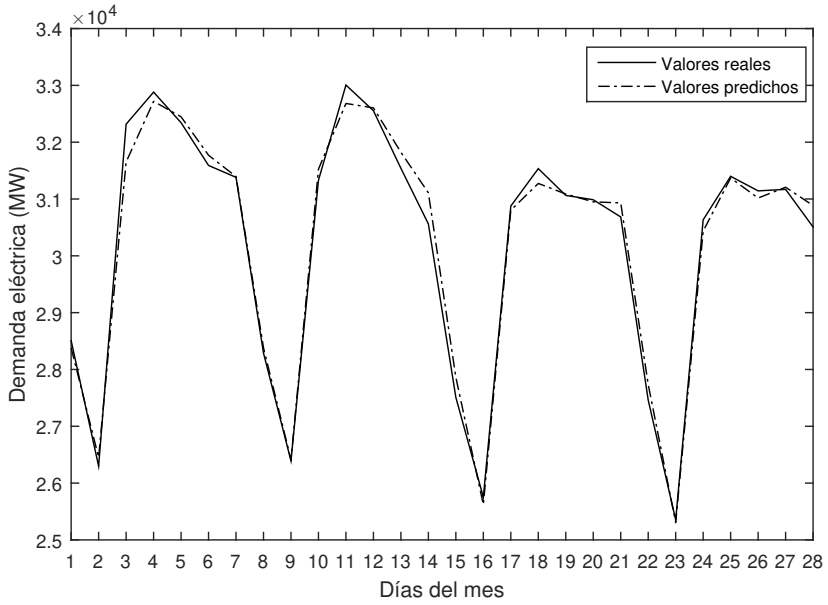


Figura 4.5: Demanda real y predicha del mes con menor MRE.

Abril del año 2016, con un 2.12 % es seleccionado como el mes con peor predicción. Se observa como la mitad de los días del mes tienen un error superior al error medio del mismo, obteniendo el peor día un 4.33 % de error. En cambio, el resto de días del mes se mantienen por debajo del error medio del mismo. Aún siendo dicho mes el que peor predicción presenta, el mejor día del mes ofrece, como en el caso anterior, una predicción por debajo del 1 % (0.89 % para ser exacto). En la Figura 4.6 se muestra la demanda real diaria para dicho mes frente a la predicción, donde se observa que la curva de valores predicha presenta el mismo comportamiento a lo largo del mes que los valores reales, aunque se presentan mayor número de imprecisiones en los picos de más alto consumo.

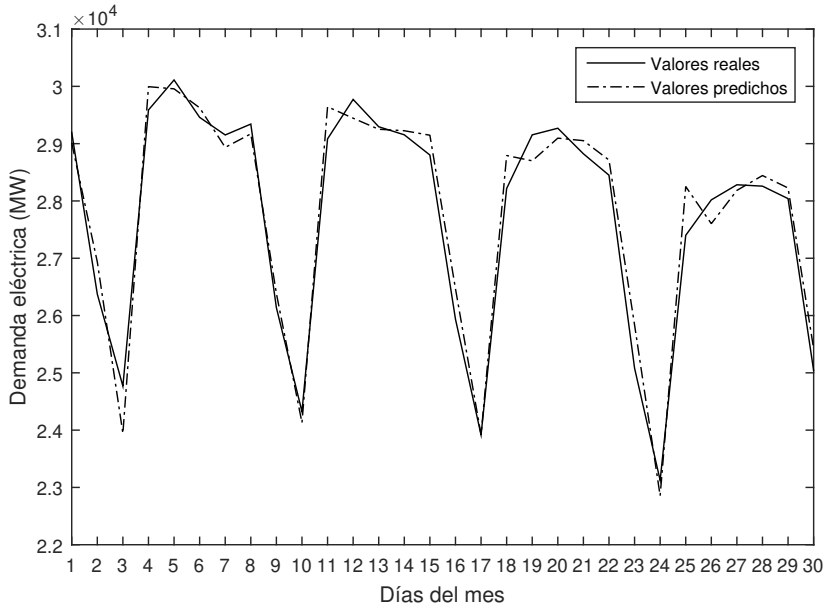


Figura 4.6: Demanda real y predicha del mes con peor MRE.

Se analizan, así mismo, los días cuya predicción arrojan tanto el mejor como el peor MRE. El mejor día obtiene menos de 0.5 % de error (0.45 % exactamente), lo que hace que tanto la curva predicha como la real fueran casi exactamente idénticas. Además destacar que dicho día se corresponde con el 17 de julio de 2015, por lo que de nuevo y como se ha comentado anteriormente, se puede observar la adaptación del algoritmo para las predicciones en los meses estivales. El peor, sin embargo, dispara el error hasta un 8.75 %, coincidiendo con un día no laborable en el territorio español, el 1 de mayo de 2014. Se comprueba que, aún siendo el peor error de todas las predicciones, el algoritmo infiere la curva de demanda correctamente, pero desplazada un número de valores.

Por último, para intentar visualizar como han sido los errores de todos los días predichos, se agrupan por meses, y se comprueba que el 75 % de todos ellos se mantienen por debajo de 2.5 %, lo cual explica el bajo porcentaje de error total conseguido en la predicción.

Aunque el error medio total conseguido es bastante bajo para este conjunto de datos, esto no indica la idoneidad del algoritmo, por lo que debe compararse con otros métodos y examinar entonces los resultados obtenidos. Así, se compara el kWNN con otras técnicas de machine learning que también han utilizado el mismo conjunto de datos, como son deep learning, regresión lineal, árboles de regresión, random forests y Gradient-Boosted Trees. Las configuraciones para cada uno de los métodos se encuentran detalladas en [81]. El kWNN resulta ser el algoritmo que menor MRE obtiene, seguido de deep learning, y siendo la regresión lineal el que peores errores ofrece.

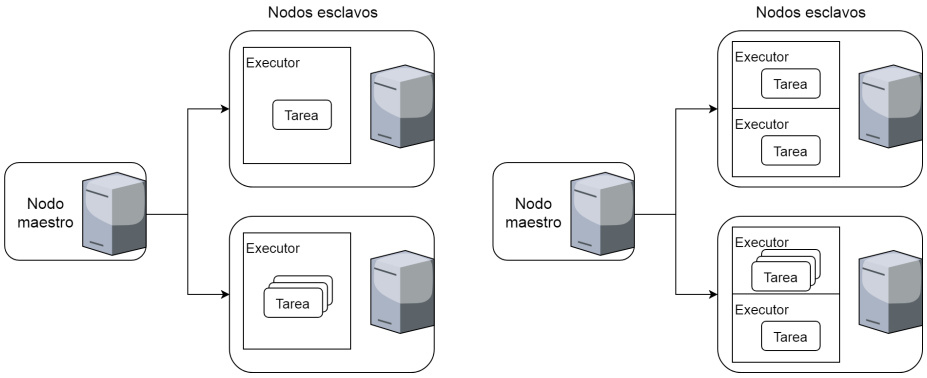
La escalabilidad del algoritmo también es analizada con diferentes configuraciones en el clúster de ordenadores disponible. Aunque Apache Spark generaliza el concepto de MapReduce y además se encarga de gestionar automáticamente los recursos de un clúster de ordenadores, evitando dichas tareas a la persona que lo utilice, hay que reseñar que cuenta con múltiples y numerosos tipos de configuraciones y parámetros, como el estudio en [32] demuestra. La mayoría de ellos se establecen de forma automática o por defecto. Pero algunos de dichos parámetros podrían tener un gran impacto en el rendimiento del clúster, y por ende, en la ejecución de un algoritmo en sí. De esta forma, para intentar aprovechar al máximo las capacidades de Spark, algunas de las configuraciones básicas del mismo son analizadas. Entre ellas el número de cores a utilizar por cada máquina dentro del clúster, el número de particiones en las que se dividirá el conjunto de datos y el número de máquinas en el clúster. Análisis más pormenorizados con la mejor configuración de todos y cada uno de los parámetros han sido objeto de estudio, por ejemplo en [32].

Es relevante comentar que, para el análisis de escabilidad, se han usado dos clúster diferentes. El primero está compuesto de dos ordenadores en el que uno de ellos actúa como maestro y el otro como esclavo. En dicho clúster se estudia el número de cores por máquina y también el número óptimo de particiones del conjunto de datos de entrada para el algoritmo. El segundo está compuesto por un mayor número de nodos, 25 para ser más exactos, donde uno de ellos actúa como máster y el resto de esclavos. En este clúster se estudia la influencia del número de máquinas en un clúster. Para ver las características exactas de dichos clústers, consultar [81].

En Spark el maestro se encarga de distribuir la información y las tareas a realizar, además de recoger los resultados y de gestionar el clúster, mientras que los esclavos (o *workers*) son los que se encargan de ejecutar dichas tareas, y por lo tanto, los que realizan la mayoría de la carga computacional. Cada tarea es efectuada dentro de un esclavo por un *executor* como se muestra en

la Figura 4.7. Por defecto, un esclavo sólo usa un *executor* (4.7 (a)) por lo que éste trabajará con todos los cores y con toda la RAM disponible. Sin embargo, Spark permite disponer de más *executors* sobre la misma máquina, dividiendo el número de cores y de RAM acorde con dicho número, y permitiendo así aceptar más tareas (4.7 (b)). De esta forma, si por ejemplo cada máquina estuviera compuesta de 12 cores y 16 GB de RAM, en la Figura 4.7 ocurriría lo siguiente:

- En (a) el único *executor* de cada máquina tiene a su disposición todos los recursos, es decir, utilizaría los 12 cores y los 16 GB de RAM.
- En (b), con dos *executors* por máquina, cada uno de ellos utilizaría 6 cores y 8 GB de RAM.



(a) Clúster con 1 executor por esclavo.

(b) Clúster con más de 1 executor por esclavo.

Figura 4.7: Diferentes configuraciones de executors en Spark.

La primera experimentación llevada a cabo en cuanto a la escalabilidad consiste en determinar la influencia del número de cores, dada la disponibilidad de memoria RAM, sobre los tiempos de ejecución del algoritmo. Se configura Spark para que el esclavo use sólo un número determinado de cores (desde 2 hasta 12) usando en cada caso toda la RAM disponible. Es importante destacar que, para esta fase de la experimentación, se configuran 2 *executors* en el nodo esclavo, excepto para la experimentación llevada a cabo con tan solo dos cores en la que se configura 1 *executor*. Un resumen de dichas configuraciones y de los resultados obtenidos se puede consultar en la Tabla 4.2. La primera conclusión que se extrae es que es preferible contar con 2 *executors* por máquina con menor memoria RAM disponible, a tener un solo *executor* que disponga de más memoria RAM. Esto hace que, con un menor número de cores aumenten los tiempos de computación, aunque en este caso cada core disponga de mayor memoria proporcionalmente. Finalmente, se concluye que es más beneficioso configurar Spark con el mayor

número de cores disponibles, aunque estos dispongan de menos RAM, lo que permite bajar los tiempos hasta casi la mitad.

Tabla 4.2: Configuraciones en el nodo esclavo para la experimentación con diferentes cores.

Número de executors	Cores por executor	RAM por executor	Cores totales	RAM total	Tiempo (horas)
1	2	14	2	14	4.8
2	2	7	4	14	3.6
2	3	7	6	14	2.95
2	4	7	8	14	2.57
2	5	7	10	14	2.3
2	6	7	12	14	2.11

La segunda experimentación realizada intenta determinar la influencia del número de particiones en las que es dividido el conjunto de datos sobre los tiempos de computación. Spark particiona automáticamente los datos, haciéndolos coincidir con el número de cores disponibles. Pero ésta división es configurable, y una correcta selección del mismo podría permitir reducir aún más los tiempos. Así, estableciendo 2 *executors* en el esclavo, cada uno de ellos con 4, 5 o 6 cores, se examinan diferentes números de particiones de los datos. Se comienza seleccionando un número superior al que automáticamente establece Spark, con la configuración óptima resultado de la experimentación anterior. En concreto 18 particiones, ya que 12 había sido el número de particiones seleccionado por Spark en el análisis previo. Después se considera el doble del anterior número de particiones, para poder encontrar un cambio en los tiempos de ejecución. Y así sucesivamente para las siguientes experimentaciones hasta llegar a 144 particiones.

Como resultado se observa que el comportamiento para cada uno de los posibles cores según la división de los datos es el mismo, alcanzándose los menores tiempos con 6 cores por *executor*, haciendo un total de 12 en uso. Se demuestra con este experimento que seleccionar un número óptimo de particiones hace que los tiempos desciendan hasta un 30 % si se comparan con el mejor tiempo obtenido en la anterior experimentación, cuando Spark dividía automáticamente el conjunto de datos. Otra conclusión extraída de este experimento es que, a su vez, un número mayor de particiones no siempre redunda en un descenso en los tiempos de ejecución, sino que, al contrario, estos pueden incrementarse.

La última experimentación consiste en analizar la influencia del número de nodos en un clúster sobre los tiempos de ejecución, una vez se han seleccionado tanto el número óptimo de cores a utilizar por máquina y el número de particiones en los que dividir el fichero de datos de entrada, según los

resultados arrojados por las anteriores experimentaciones.

Se consideran clústers con número de máquinas desde 1 hasta 24 esclavos. Es importante reseñar que, a su vez, se realizan dos experimentaciones para cada tipo de clúster analizado. Una en la cual cada máquina cuenta con un único *executor* con todos los cores y memoria RAM disponibles para éste. La otra experimentación se realizó estableciendo dos *executors* por máquina, dividiendo a su vez los cores y la memoria RAM entre cada uno de ellos. La conclusión más obvia a los resultados de los experimentos es que, a mayor número de esclavos en el clúster, menor tiempo de computación, hasta 16 máquinas. Pero el otorgarle el máximo número de máquinas no resulta en una mayor reducción de tiempos, sino todo lo contrario, estos aumentan. Es decir, para obtener los mejores tiempos en la ejecución de un algoritmo, un clúster no debe contar con el mayor número posible de máquinas, con el susodicho ahorro de contratación de recursos que esto supondría. Es más, con 8 máquinas se obtienen unos resultados suficientemente competitivos si se tuviera que determinar el tamaño exacto del clúster a contratar.

Por otro lado, se observa que si contamos con un clúster pequeño, de entre 1 a 4 esclavos, es preferible configurar Spark con dos *executors* por máquina que con uno. Sin embargo, si se cuenta con un clúster con mayor número de máquinas, no se observa una diferencia sensible en tiempos de ejecución entre configurar 1 o 2 *executors* por máquina.

Como conclusión final a todas las experimentaciones realizadas, y comparando los mejores tiempos obtenidos con la versión preliminar del algoritmo en [83] (53 minutos) con el mejor tiempo obtenido en [81] (64 minutos), a pesar de subir ligeramente este último, se han tratado más del doble de datos que en el primero y además obteniendo mucho mejores resultados en cuanto a la predicción gracias a la selección óptima de parámetros del algoritmo.

4.2.2. Resultados con algoritmo multivariante

Aunque la predicción en la anterior experimentación alcanza unos niveles de error realmente bajos para el conjunto de datos analizado, puede encontrarse una serie temporal para la cual la predicción de su futuro no dependa tanto del comportamiento de ésta en el pasado, sino de otras variables exógenas a la misma.

Es por ello que se desarrolla el algoritmo multivariante MV-kWNN, como se ha comentado en la Sección 3.3 del Capítulo 3. Una vez el método ha sido implementado y antes de realizar cualquier tipo de predicción, es necesario llevar a cabo un estudio de la viabilidad del método. Dicho estudio consiste en descubrir la correlación mínima que debe darse entre las series temporales analizadas, más en concreto, la correlación mínima entre la variable exógena y los valores futuros de la variable objetivo. De esta forma, si una serie temporal no es lo suficientemente influyente sobre los valores futuros de la

otra (no alcanza cierto nivel de correlación), las predicciones no serán mejores que en la versión univariante, es decir, que no serán mejores que estudiando sólo el pasado de la propia serie objetivo.

Este análisis se realiza creando series temporales sintéticas de dos tipos:

1. 5 series temporales con distintos niveles de correlación con sus propios valores futuros (h) que representan las variables objetivo.

2. 5 series temporales más para cada una de las anteriores, que representan las variables exógenas, dando lugar a un total de 25 series. Estas nuevas series presentan distintos niveles de correlación con el futuro de las variables objetivo.

Se mide el MRE obtenido en la predicción de las 25 experimentaciones extrayéndose las siguientes conclusiones (para un mejor entendimiento, se recomienda ver la Figura 4.8):

- Como es de esperar, a medida que aumenta la correlación entre el pasado de la variable exógena y el futuro de la serie objetivo, se obtienen mejores predicciones.

- De igual modo, a medida que la correlación entre valores pasados y futuros de la serie objetivo aumenta, también disminuye el error en las predicciones.

- En general, siempre que la correlación entre el pasado de la variable exógena y el futuro de la serie objetivo es mayor o igual a la correlación entre valores pasados y futuros de la serie objetivo, se obtienen mejores predicciones con el algoritmo multivariante que con el algoritmo univariante. Así, por ejemplo, si la correlación entre valores pasados y futuros de la serie objetivo fuera 0 y se contara con una variable exógena con correlación entre pasado de la variable exógena y futuro de la serie objetivo superior o igual a 0.25, se obtendrían mejores resultados ejecutando el algoritmo multivariante. Mientras, si la correlación entre valores pasados y futuros de la serie objetivo fuera 1 o cercano a 1, sólo una variable exógena con correlación entre pasado de la variable exógena y futuro de la serie objetivo 1 o similar a 1 podría hacer que la predicción multivariante mejorara a la predicción univariante.

Enumeradas las anteriores conclusiones, cabe destacar que si la correlación entre valores pasados y futuros de la serie objetivo es mayor que la correlación entre pasado de la variable exógena y futuro de la serie objetivo, el algoritmo multivariante obtiene predicciones levemente peores en lo que a error se refiere, lo que demuestra la robustez del método multivariante aún no escogiendo las variables exógenas adecuadas para realizar las predicciones.

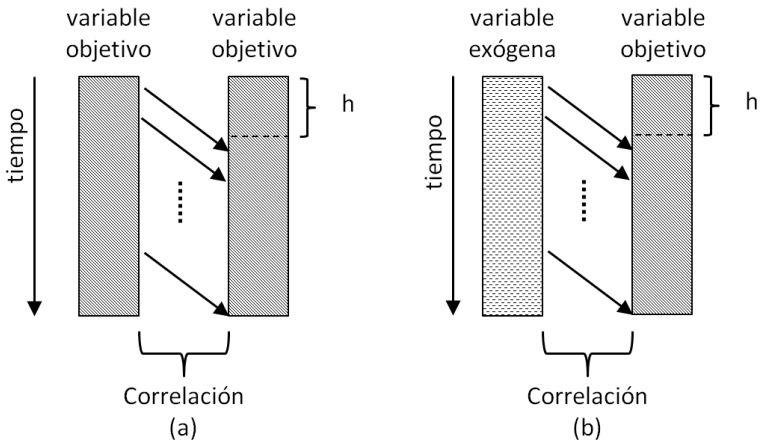


Figura 4.8: Ejemplo de correlaciones entre una variable objetivo y una variable exógena.

Se lleva a cabo entonces la predicción de un caso real. Se utiliza nuevamente la serie temporal de demanda eléctrica ya usada para la versión univariante, y además se añade la serie temporal con los precios de la electricidad. Esta última está compuesta por los datos comprendidos entre los años 2007 y 2016, es decir de igual longitud que la demanda, pero medida cada hora en lugar de cada 10 minutos, como se puede apreciar en la Figura 4.9. Cabe destacar que dichos precios fueron extraídos en ficheros que contenían sólo los datos de un día, lo que suponía descargar más de 3265 ficheros y unirlos posteriormente. Además, se preprocesaron dichos datos ya que algunos expresaban el precio en Cent/kWh mientras que otros lo hacían en Euro/MWh, por lo que hubo que unificarlos y finalmente expresarlos todos en Euro/MWh.

Los datos tanto de la demanda como del precio son normalizados por el propio algoritmo, dada la diferente escala de cada una de las series temporales. El MV-kWNN los desnormaliza una vez realizadas las predicciones para calcular los errores cometidos.

```
2007-01-01 02:00,48.00
2007-01-01 03:00,33.17
2007-01-01 04:00,26.83
2007-01-01 05:00,24.24
2007-01-01 06:00,22.45
2007-01-01 07:00,20.22
2007-01-01 08:00,16.00
```

Figura 4.9: Conjunto de datos de precios de electricidad en España.

Tomándose la demanda eléctrica como variable objetivo y los precios como variable exógena, se obtiene que la correlación entre valores pasados y futuros de la serie objetivo es mayor que la correlación entre pasado de la

variable exógena y futuro de la serie objetivo. De igual forma ocurre si la variable objetivo son los precios y la variable exógena la demanda. Cabe esperar entonces errores ligeramente superiores a los obtenidos por el algoritmo univariante kWNN.

El parámetro del algoritmo w es fijado a 1 día y se considera un horizonte h de 4 horas. El número de vecinos cercanos k es 4 en este caso. Esto es debido a que la elección de un número de vecinos u otro no hace que aumente o disminuya de forma considerable el error, como habían demostrado los análisis sobre el tamaño óptimo de este parámetro explicado en el Apartado 4.2.1 de este Capítulo. Aún así, un estudio parecido es llevado a cabo para el algoritmo multivariante, demostrando lo comentado anteriormente.

Analizando los resultados obtenidos en la predicción de la demanda, se obtiene un error superior al método de predicción univariante, como era de esperar dadas las correlaciones entre las series, como se explicó anteriormente. Sin embargo, cabe reseñar que éste sólo se eleva algo más de un 1 %, alcanzando finalmente un MRE de 2.72 %. Esto confirma la robustez del método como también señaló el análisis de correlaciones entre series. Es importante destacar que, al igual que en el algoritmo univariante, se predicen el 30 % de los valores del conjunto de datos, lo que supone un total de 2 años y 10 meses.

Visualizando todos los errores cometidos en la predicción, agrupados por meses, para intentar detectar posibles comportamientos anómalos en el algoritmo, se observa como la mayoría de errores, concretamente el 75 %, son menores que 5.5 %.

Como se puede ver en la Tabla 4.3, analizando con más detalle el error obtenido mensualmente, se observa como las peores predicciones se suelen dar en los meses invernales, quedando en la mayoría de los casos el error en dichos meses por encima del error total. Es más, diciembre y enero, representan los meses con peor error en los años 2013 y 2016 respectivamente. Para el año 2014 la peor predicción se corresponde con el mes de abril, mientras que para 2015 se corresponde con febrero. Por otro lado, y al contrario de como podría pensarse, los meses de verano no sólo no alcanzan las tasas mayores de errores en la predicción sino que en algunos años representan la mejor predicción, como sucede en los meses de agosto y julio en 2013 y 2014. Noviembre y mayo también son los meses con menor error para los años 2015 y 2016 respectivamente.

Tabla 4.3: MRE de las mejores y peores predicciones de la demanda cada año.

Año	2013		2014		2015		2016	
Mejor mes	Ago.	2.39 %	Jul.	2.17 %	Nov.	2.11 %	May.	2.20 %
Peor mes	Dic.	4.36 %	Abr.	3.57 %	Feb.	3.06 %	Ene.	3.10 %

Como en anteriores experimentaciones, los meses con el menor y el mayor error son analizados más detalladamente a continuación.

Noviembre de 2015 es el mes con menor tasa de error de todas las predicciones. Se observa como sólo 12 días del mes tienen un error mayor que el error total del mes, obteniéndose como error máximo, correspondiente al peor día, un 5.19 %. El resto de días tienen errores cercanos dicho error total del mes o por debajo del mismo, alcanzándose un error casi nulo (0.55 %) en el día con mejor predicción. En la Figura 4.10 se muestra la demanda real diaria para dicho mes frente a la predicción, donde se observa como las curvas presentan una gran similitud, prediciendo especialmente bien las bajadas en la demanda, pero no siendo tan idénticas en los cambios que se producen en los días con mayor demanda.

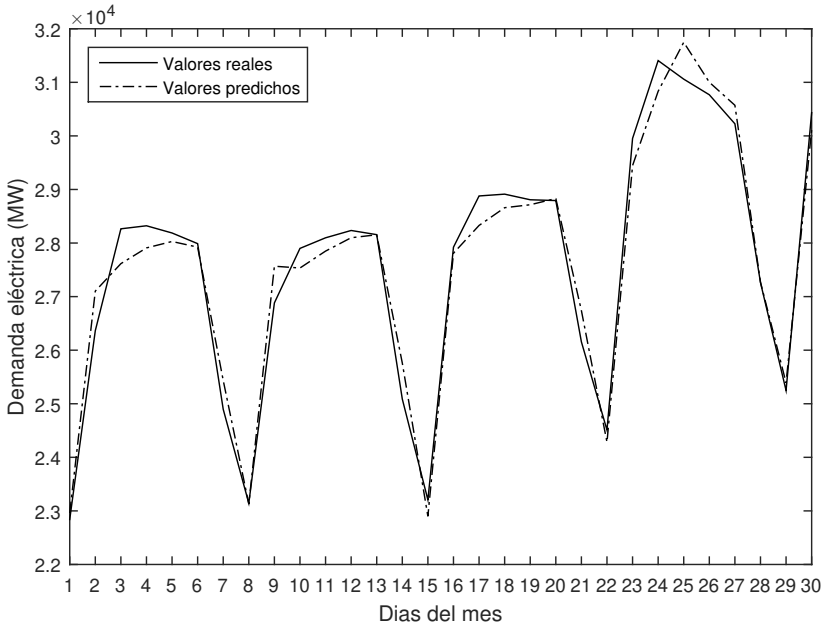


Figura 4.10: Demanda real y predicha del mes con mejor MRE.

Diciembre del año 2013, con un 4.36 % de tasa de error, resulta ser la peor predicción. En este mes se observa que, aun no contando con un gran número de días con error por encima del error total del mes, sí se obtienen días con una tasa de error bastante alta, llegando a alcanzar 6 días un error por encima de 7 %. Aún siendo el mes que peor error ofrece, el día con menor error dentro de dicho mes alcanza un 1.20 %. En la Figura 4.11, donde se muestra la demanda real del mes frente a la predicha, se puede comprobar que ambas curvas siguen el mismo comportamiento, pero para el día 7 el algoritmo predice una bajada en la demanda que no resulta ser tan baja. Lo mismo ocurre con la subida que el algoritmo predice para el día 30, pero que

no llega a ser tan pronunciada. Estas diferencias son las que hacen encarecer el error del mes, ya que es en ambos días cuando se alcanzan las mayores tasas de MRE.

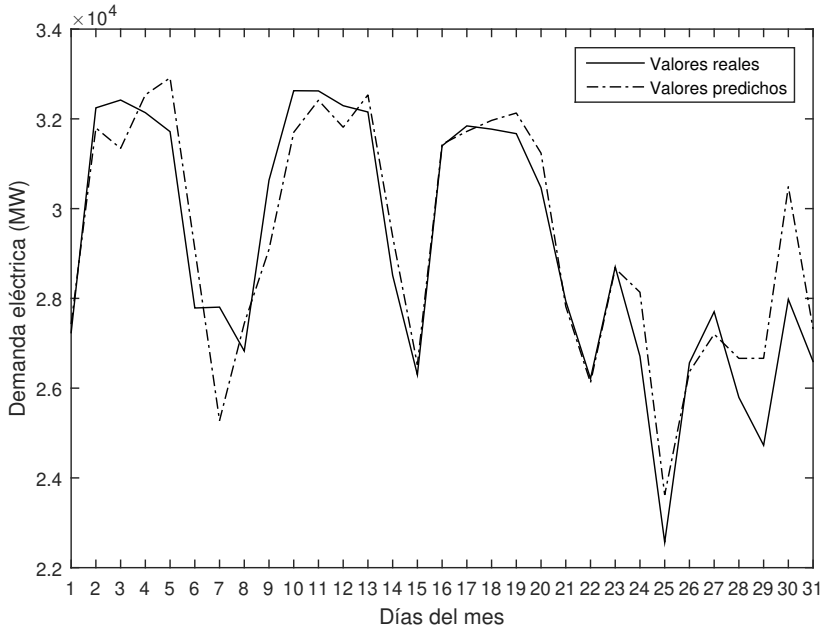


Figura 4.11: Demanda real y predicha del mes con peor MRE.

Se analizan así mismo los días cuya predicción arrojan el mejor y el peor error. El mejor día obtiene un error cercano a 0 % (0.40 %), el 23 de julio del año 2015, de nuevo un día del periodo estival, lo que confirma la idoneidad del algoritmo para este tipo de meses. El peor día por otro lado, alcanza un 11.68 % correspondiéndose con el 10 de febrero de 2014. Se comprueba que, aún teniendo un error alto, el algoritmo predice adecuadamente el comportamiento de la curva, pero desplazado un número de valores. Dicho día de febrero, se corresponde con un lunes del mes. Como día laborable, la curva de demanda comienza a ascender rápidamente desde las 7:00 AM hasta las 10:00 AM que alcanza uno de sus picos máximos. El algoritmo, sin embargo, predice dicho día como si hubiera sido un día no laborable, pues el ascenso en la demanda no se empieza a producir hasta las 10:00 AM alcanzando su pico más alto a la 01:00 PM. El comportamiento, aunque casi idéntico, está desplazado 3 horas, y la predicción además ofrece una menor cantidad de MW. Después, la demanda real decrece hasta las 04:00 PM, donde se mantiene, para volver a ascender entorno a las 07:00 PM, alcanzando el pico máximo a las 10:00 PM. La predicción dada por el algoritmo también detecta esa bajada hasta las 04:00 PM, pero después, en lugar de mantenerse, desciende levemente hasta las 08:00 PM en lugar de hasta las 07:00 PM como hace la

demanda real. Finalmente, la predicción asciende hasta la misma hora que la demanda real, las 10:00 PM, pero ofreciendo un menor valor en MW que la demanda real.

A continuación, se analizan también las predicciones realizadas de los precios de la electricidad. La tasa de error cometida en la predicción de esta serie temporal es superior a la demanda como es de esperar, debido a la no estacionariedad de la misma, con subidas y bajadas bruscas de precio llegando a contar con un número importante de medidas iguales a 0. 10.67 % es el error obtenido. Visualizando los errores de cada una de las predicciones realizadas agrupadas por meses, no se detecta ninguna anomalía reseñable, pero destaca diciembre de 2013 como el mes que presenta más porcentajes de errores altos. Dicho mes es analizado más en detalle.

En la Tabla 4.4 pueden verse los meses que mejor y peor MRE obtienen para cada año. De nuevo, y como hemos destacado en los anteriores análisis, se comprueba la adecuación del algoritmo para predecir los meses estivales, o cercanos a verano, de esta serie temporal, dado que los meses con mejores predicciones se dan en junio (2015) y agosto (2014) y septiembre (2013). También de nuevo se observa que los meses de invierno son los que peor predice el algoritmo, ya que en esta nueva serie también ha sido cuando ha alcanzado los errores más pronunciados. Más en concreto, los meses de diciembre en 2013 y enero en 2016. El mes de febrero, por otro lado, es el que mayor tasa de error obtiene para los años 2014 y 2015.

Tabla 4.4: MRE de las mejores y peores predicciones de los precios cada año.

Año	2013		2014		2015		2016	
Mejor mes	Sep.	9.67 %	Ago.	6.98 %	Jun.	6.46 %	May.	7.69 %
Peor mes	Dic.	22.01 %	Feb.	18.91 %	Feb.	12.21 %	Ene.	11.46 %

Nuevamente, los meses concretos con la mejor y la peor predicción son analizados.

Junio de 2015 alcanza el menor MRE. Se observa como el error de 13 días del mes exceden el error total del mes en sí, con 8 de ellos superando el 8 % de error y llegando a situarse el mayor de todos con un 11.19 %. Por otro lado, el error de 6 días del mes desciende del 4 % llegando el menor de todos a 2.78 % de error. En la Figura 4.12 se muestra el precio real diario para dicho mes frente a la predicción realizada. El algoritmo predice adecuadamente el comportamiento que tiene la curva de precios, dando una casi total coincidencia en la bajada de los precios, pero difiriendo levemente en los picos de precios altos.

Diciembre de 2013 alcanza el peor error. Este mes no cuenta con un gran número de días con error superior error total del mismo, pero 6 de ellos tienen errores superiores al 30 %, lo que hace que el error total del mes suba. De hecho, el error de dos días alcanza casi el 45 %, siendo estos días los que

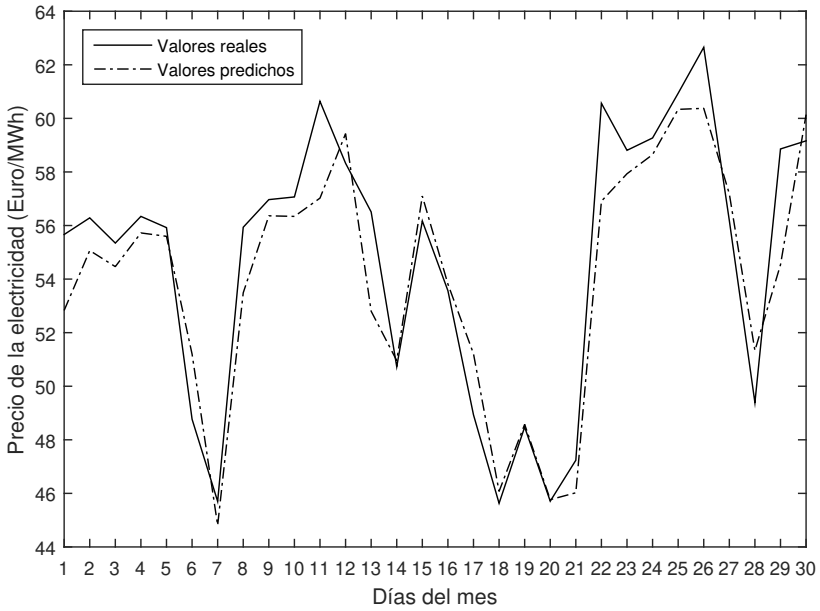


Figura 4.12: Precio real y predicho del mes con mejor MRE.

peor predicción arrojan del total de predicciones realizadas. Uno de los días se analizará más en detalle. Con respecto al resto de días, resaltar que 6 días del mes obtienen un error inferior a 12.5 %, siendo el menor de todos 6.76 %. En la Figura 4.13 se ha representado los valores de los precios reales diarios frente a los predichos. En este caso se comprueba que el algoritmo predice un comportamiento bastante similar a la curva real de datos, acertando incluso en la bajada pronunciada de precios que se produce entre los días 23 y 25. Pero para el resto de valores se encuentra desplazado un número que podría considerarse considerable, lo que hace que suban las tasas de error.

Nuevamente, se analizan los días cuya predicción ofrece tanto el mejor como el peor MRE. El mejor día, el 16 de abril de 2014, obtiene una predicción bastante baja, de tan sólo un 2.02 % de error, en la que el algoritmo acierta incluso en las bajadas y subidas pronunciadas de los precios. El día con el peor error, el 20 de diciembre de 2013 sin embargo, se dispara hasta un 45.29 % como se ha comentado anteriormente al mencionar el mes con el peor MRE, ya que pertenece a dicho mes. Se observa como el algoritmo imita la curva de los precios, pero el desplazamiento con respecto a la curva de precios reales es considerable, incrementando el error final del mes. Sobre todo destacar que el algoritmo predice la bajada de precios, coincidiendo incluso hasta en la hora en la que se alcanza el valor más bajo para luego volver a subir, pero no hasta el valor 0 como en la realidad ocurre, siendo entonces esas diferencias las que más penalizan la predicción final.

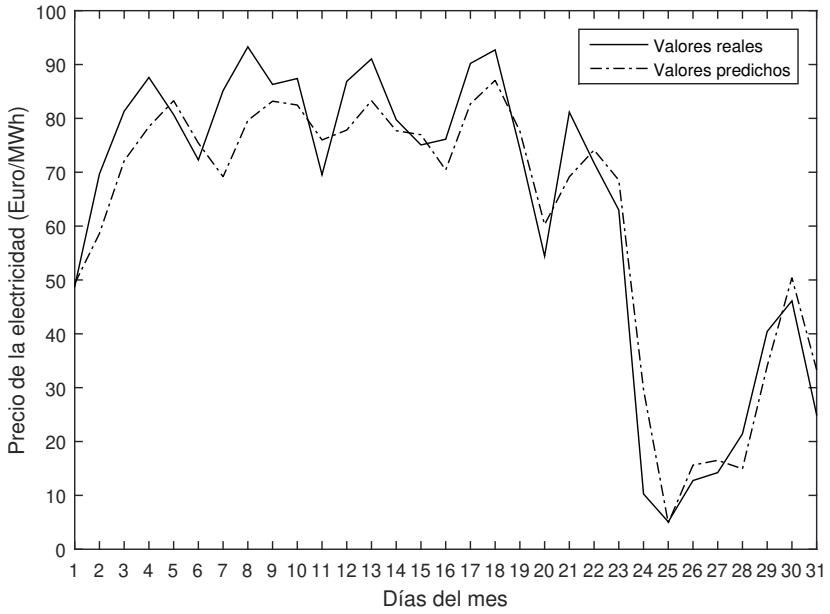


Figura 4.13: Precio real y predicho del mes con peor MRE.

Aunque los resultados alcanzados tras ejecutar el MV-kWNN sobre las series temporales de demanda eléctrica y precio de la electricidad son bastante buenos, se necesita comparar con otros métodos de predicción multivariante para así confirmar la validez del algoritmo. Así, se han seleccionado métodos clásicos adaptados para analizar variables exógenas, como son ARX, ARMAX y ARIMAX. Algunos algoritmos de machine learning también tienen disponible su versión multivariante, como por ejemplo redes neuronales (MV-ANN) y random forests (MV-RF). Lo primero que se realiza antes de ejecutar los algoritmos es adaptar los conjuntos de datos, ya que aunque MV-kWNN permita series temporales con diferentes intervalos, los algoritmos seleccionados para la comparación no lo contemplan. La serie temporal de la demanda eléctrica pasa a ser horaria en lugar de cada 10 minutos, con la consiguiente reducción en el tamaño del conjunto de datos a manejar.

Además, no sólo se compara el MRE obtenido en las predicciones, sino que se utilizan otras 3 nuevas medidas de precisión: el error absoluto medio (MAE), el bias (BIAS) y el error cuadrático medio (RMSE). Las configuraciones establecidas para cada método pueden encontrarse en [82].

Una vez realizadas todas las experimentaciones con los métodos propuestos, se analizan los resultados obtenidos. Se confirma que el MV-kWNN mejora a los demás, en cuanto a MRE, MAE y RMSE se refiere, tanto en la demanda como en las predicciones del precio. Si nos centramos en la demanda, el MRE obtenido por el MV-kWNN es ligeramente inferior a la versión

multivariante de las redes neuronales y a la versión multivariante de randon forests, siendo en este caso ARX el que mayor MRE presenta, un 66 % peor que el obtenido por el MV-kWNN. En lo que a BIAS se refiere, todos los métodos obtienen un valor negativo, excepto el ARIMAX.

En la predicción de los precios ocurre algo similar, MV-kWNN se confirma como el mejor algoritmo, en términos de MRE, MAE y RMSE. Si nos centramos en el MRE, se comprueba como los errores son muy similares entre el MV-kWNN, el MV-RF y el MV-ANN, segundo y tercero mejores respectivamente. El que mayores errores obtiene es de nuevo ARX, un 22 % superior a MV-kWNN. En lo que a BIAS se refiere, los métodos MV-kWNN, MV-ANN y ARMAX obtienen valores negativos mientras que MV-RF, ARX y ARIMAX obtienen valores positivos. Además, MV-kWNN, MV-RF y ARMAX obtienen BIAS muy cercano a 0.

Aunque MV-kWNN se alza con los mejores resultados en las predicciones realizadas sobre estas dos series temporales, la diferencia con los siguientes métodos es muy pequeña, por lo que se realiza un test estadístico para evaluar la relevancia del nuevo algoritmo multivariante propuesto. Se utilizan 5 series temporales sintéticas y se combinan entre sí para llevar a cabo la predicción de todas ellas, dando lugar a 10 experimentaciones más para cada método evaluado. El MRE es la medida de precisión elegida para evaluar las predicciones realizadas, y el resultado al aplicar un test de Friedman sobre éstas otorga la primera posición en el ranking al MV-kWNN, confirmando la idoneidad del algoritmo propuesto.

Por último, se realiza un nuevo estudio de escalabilidad sobre esta versión multivariante, para comprobar la adaptación del algoritmo a series temporales Big Data. Para ello se multiplica el tamaño original de los conjuntos de datos hasta un máximo de 200, lo que hace que estás últimas series temporales sintéticas estén compuestas por más de 16 millones de registros cada una. Se analiza los tiempos de cálculo de una predicción del algoritmo propuesto y se compara con los tiempos obtenidos por los demás métodos para realizar la misma tarea. Todas las experimentaciones son llevadas a cabo sobre un clúster de 3 ordenadores, 1 actuando como maestro y los otros dos como esclavos. Es necesario resaltar que las implementaciones de los métodos encontrados (ARX, ARMAX, ARIMAX, MV-ANN y MV-RF) no tienen la posibilidad de ejecutarlos en un clúster de máquinas, por lo que los tiempos reportados son sobre una única máquina. Cuando el tamaño de los datos es el original, los algoritmos que obtienen mejor tiempo son el MV-ANN y el MV-RF, seguidos muy de cerca por el MV-kWNN. Esto podría deberse a que, como se comentó anteriormente, MV-kWNN analiza un conjunto mayor de datos, al poder tomar diferente series temporales con distintos intervalos de tiempo, mientras que MV-ANN y MV-RF cuentan con series temporales horarias y por tanto más pequeñas. ARMAX por otro lado, reporta los mayores tiempos de computación.

Realizando pruebas con tamaños mayores de conjuntos de datos, se encuentra un problema con los métodos ARX, ARMAX, ARIMAX, MV-ANN y MV-RF. Al no estar preparados para ejecutarse en un clúster de ordenadores y no estar adaptados para Big Data, al intentar llevar a cabo las predicciones con el segundo conjunto de prueba, esto es, multiplicando por 10 el tamaño del conjunto de datos original, es imposible recabar los tiempos de dichos algoritmos, eternizándose estos sin posibilidad de conocer cuando terminarían. Por lo que se descarta este análisis para dichos métodos. El MV-kWNN, sin embargo, no presenta dicho problema. Se compara entonces los tiempos obtenidos por el algoritmo al ejecutarse sobre un clúster con un esclavo y un clúster con dos esclavos. Los resultados alcanzados demuestran que los tiempos de ejecución no son muy diferentes en un clúster u otro con tamaños del conjunto de datos hasta 50 veces más grandes. Pero cuando los datos comienzan a ser ya masivos, esto es, con tamaños del conjunto de datos 100 y 200 veces más grandes, los tiempos se disparan de forma exponencial para el clúster con un esclavo, mientras que para el clúster con dos esclavos suben sólo de forma leve y linealmente. Esto confirma la adecuación del MV-kWNN para ejecutarse con series temporales Big Data y resalta la importancia de contar con un clúster de máquinas con varios esclavos a la hora de llevar a cabo la computación.

Capítulo 5

Conclusiones generales y trabajos futuros

El final de una obra debe hacer recordar siempre el comienzo.

Joseph Joubert.

Resumen: En este capítulo se recogen las principales conclusiones alcanzadas con la consecución de este trabajo de tesis, además de comentar las posibles líneas de trabajo futuro que se presentan.

El afán de conocimiento y la predicción del futuro han sido desde el principio de los tiempos dos de las mayores obsesiones del ser humano. Para intentar alcanzarlas o darles respuesta, nos hemos dedicado a recabar toda la información posible, almacenando todo posible dato que se pudiera registrar para, posteriormente, poder analizarlos. En la actualidad, acumular y guardar datos no supone ningún problema dado el bajo coste, a nivel económico y computacional que ello supone. Esto ha hecho que registremos cualquier dato posible de nuestro día a día, desde los latidos de nuestro corazón, el tráfico o el consumo eléctrico realizado. Es en este contexto donde surgen las Smart Cities, almacenando todo tipo de información y registrando cualquier posible cambio para, posteriormente, mejorar la vida de los ciudadanos que las habitan. Pero es tal el volumen de datos generados en los últimos años, que se ha hecho imposible no ya sólo extraer conocimiento de toda ella sino incluso procesarlas. Las técnicas clásicas de análisis, así como las de machine learning, que tan buenos resultados alcanzaron con tamaños de datos cuantificables, necesitan ser revisadas para adaptarse al nuevo contexto del Big Data.

Este trabajo de investigación ha intentado dar respuesta a esta problemática, desarrollando un nuevo algoritmo, basado en vecinos cercanos, para

predecir series temporales Big Data. Para ello, apoyándose en un nuevo framework de programación y análisis de datos, como en este caso Apache Spark para llevar a cabo computación distribuida, se desarrolló el algoritmo kWNN. La implementación en Spark de dicho algoritmo supuso un reto, al plantear el framework una nueva forma de programar distinta a las clásicas conocidas, con el objetivo de lanzarse en un clúster de ordenadores y sacar el máximo provecho a su computación distribuida. Además, se trabajó con el lenguaje de programación nativo de Spark, Scala, un lenguaje de programación funcional del que no se tenía conocimientos previos. La primera implementación obtuvo resultados prometedores, tanto en tiempos de ejecución como en la precisión de las predicciones. Se usaron para probar el algoritmo series temporales reales de Smart Cities, con el consumo eléctrico medido cada 15 minutos durante 3 años de dos edificios de una universidad pública.

El algoritmo desarrollado contaba con una serie de parámetros, como eran el número de valores pasados que hay que usar (w) para predecir los h valores siguientes y el número de vecinos cercanos k a considerar para la predicción. Si dichos parámetros se escogían adecuadamente, la predicción podría mejorar considerablemente, por lo que para la siguiente versión del algoritmo se añadió esta mejora, aparte de perfeccionar la propia implementación. Usando una serie temporal de Smart Cities diferente y de mayores proporciones, en este caso el consumo eléctrico a nivel nacional medido cada 10 minutos durante 9 años, se probó la nueva versión del algoritmo. Los resultados obtenidos fueron de un error inferior al 2 % al predecir casi 3 años de la serie temporal. Analizando más en detalle el comportamiento en la predicción del algoritmo se observó como éste predecía de forma más adecuada los meses que en principio podrían considerarse los más complicados, esto es, los meses estivales, dada la cantidad de personas de vacaciones durante esta época, lo que resaltaba la adaptabilidad del algoritmo. Por otro lado, los meses en los que más elevado error se observó fueron los meses correspondientes al periodo invernal, siendo casi siempre diciembre el que mayor error obtenía.

Se comparó el error de la predicción del conjunto de test con otras técnicas de machine learning que también habían usado el mismo conjunto de datos, como deep learning o árboles, resultando el kWNN el que menor error obtenía. Ello demostraba la idoneidad del algoritmo así como también justificaba las mejoras introducidas. Un análisis sobre un clúster de ordenadores fue también realizado para intentar conseguir los menores tiempos de computación. Varias conclusiones fueron extraídas. La primera fue que era preferible contar con máquinas con el mayor número de cores posible, aunque estos tuvieran menor memoria RAM proporcionalmente, que contar con pocos cores con mayor memoria RAM disponible. La segunda conclusión fue que el número de divisiones en el cual el conjunto de datos debe dividirse tiene que ser escogida adecuadamente, ya que aunque Spark los divide

de forma automática, esto no implica que se obtengan los mejores tiempos. Seleccionar bien este parámetro hace disminuir los tiempos de computación sensiblemente. Por último se analizó la influencia del número de máquinas esclavas en el clúster, demostrándose por un lado que al contar con más de dos los tiempos descendían, aunque, por otro lado, no siempre el contar con el mayor número posible hace que se obtengan los mejores tiempos, sino que al contrario, hace que estos se eleven.

Pero en muchas ocasiones una serie temporal no está influenciada sólo por su propio pasado, sino que también lo está por el comportamiento de otras series temporales exógenas. Es por ello que, una vez vistas las cualidades que presentaba el algoritmo en versión univariante, se desarrolló una nueva versión para que tuviera en cuenta no una sino varias series temporales y al mismo tiempo predijera todas ellas. Se realizó entonces un análisis de correlación entre series para identificar cuando el añadir una variable exógena al problema mejoraría la predicción o la empeoraría. El resultado determinó que si el futuro de una serie temporal está más correlacionado con una variable exógena que con su propia serie, la predicción con el algoritmo multivariante sería mejor que con el univariante. En general, se determinó que el algoritmo era lo suficientemente robusto para que, aunque la serie exógena se añadiera al problema y no estuviera lo suficientemente correlacionada, los errores en la predicción sólo aumentaran levemente. Así, tomando dos series temporales reales, la demanda eléctrica en España y los precios de la electricidad a nivel nacional en los últimos 9 años, se llevó a cabo la predicción. Los resultados mostraron tasas de error muy bajas, de nuevo mostrando el algoritmo mejor adaptación y por tanto mejores predicciones en los meses estivales, siendo a su vez los meses de invierno los que alcanzaban mayor error. La precisión del algoritmo fue comparada con otros algoritmos multivariantes, como las versiones adaptadas de los métodos clásicoss de Box y Jenkins, y redes neuronales y random forests, resultando el MV-kWNN el mejor posicionado en los tests estadísticos realizados. Además, se analizó el comportamiento del algoritmo con series temporales de tamaño considerable, hasta 200 veces mayor que las tratadas en los experimentos, demostrando como los demás métodos no eran capaces de gestionar tal cantidad de información mientras que el MV-kWNN ofrecía escalabilidad lineal, reafirmando la adaptación del algoritmo a contextos Big Data.

Como trabajos futuros, se proponen analizar en más detalle los meses de invierno e intentar combinar el algoritmo con otro método que pudiera predecir mejor dichos meses, creando una solución *ensemble* que mejorara más aún los resultados ofrecidos.

Otra línea de investigación que se propone es adaptar los algoritmos para que puedan efectuar análisis y predicciones en tiempo real, para así, al mismo tiempo que los dispositivos IoT recogen las medidas, tomar las decisiones oportunas y actuar en consecuencia.

Parte II

Publicaciones

Capítulo 6

Trabajos de investigación publicados

El genio comienza las grandes obras,
pero sólo el trabajo las acaba.

Petrus Jacobus Joubert.

Resumen: En este capítulo se muestran los trabajos que comprenden esta tesis por compendio de artículos. Primero se adjunta un artículo de congreso internacional que supuso el inicio de esta tesis. Seguidamente, se encuentran los dos artículos publicados en revista, el primero con la versión univariante del algoritmo para predicciones de series temporales Big Data y el segundo con la versión multivariante del mismo. Así mismo, también se pueden encontrar en este capítulo otros artículos de investigación en los que se ha colaborado, aunque no se contabilizan para la tesis por compendio de artículos. Concretamente, un artículo de congreso internacional aplicando un algoritmo k-means distribuido sobre series temporales Big Data y un artículo publicado en revista con un algoritmo *ensemble* que combinaba random forests, un árbol de regresión y GBT sobre también series temporales Big Data.

6.1. A nearest neighbours-based algorithm for big time series data forecasting

El artículo en congreso internacional publicado es el siguiente:

- R. L. Talavera-Llames, R. Pérez-Chacón, M. Martínez-Ballesteros, A. Troncoso, and F. Martínez-Álvarez. A nearest neighbours-based algorithm for big time series data forecasting. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 174-185. 2016.

A Nearest Neighbours-Based Algorithm for Big Time Series Data Forecasting

Ricardo L. Talavera-Llames[1], Rubén Pérez-Chacón[1], María Martínez-Ballesteros[2], Alicia Troncoso[1], and Francisco Martínez-Álvarez[1(✉)]

[1] Division of Computer Science, Universidad Pablo de Olavide, 41013 Seville, Spain
{rltallla,ali,fmaralv}@upo.es, rpercha@alu.upo.es
[2] Department of Computer Science, University of Seville, Seville, Spain
mariamartinez@us.es

Abstract. A forecasting algorithm for big data time series is presented in this work. A nearest neighbours-based strategy is adopted as the main core of the algorithm. A detailed explanation on how to adapt and implement the algorithm to handle big data is provided. Although some parts remain iterative, and consequently requires an enhanced implementation, execution times are considered as satisfactory. The performance of the proposed approach has been tested on real-world data related to electricity consumption from a public Spanish university, by using a Spark cluster.

Keywords: Big data · Nearest neighbours · Time series · Forecasting

1 Introduction

Recent technological advances have led to a rapid and huge data storage. In fact, 90 % of existing data in the world have been generated in the last decade. In this context, the improvement of the current data mining techniques is necessary to process, manage and discover knowledge from this big volume of information. The modern term big data [10] is used to refer to this evolution. Sensor networks, typically associated with smart cities, are one of the main sources of big data generation and can be found in diverse areas such as energy, traffic or the environment.

The popular MapReduce paradigm [4] has been recently proposed by Google for big data parallel processing. This paradigm has been widely used by Apache Hadoop [15], which is an open source software implemented in Java and based on a distributed storage system called Hadoop Distributed File System (HDFS). However, the limitations of the MapReduce paradigm to develop iterative algorithms have promoted that other proposals emerge, such as Apache Spark [6]. Apache Spark is also an open source software project that allows the multi-pass computations, provides high-level operators, uses diverse languages (Java, Python, R) in addition to its own language called Scala, and finally, offers the machine learning library MLlib [5].

F. Martínez-Álvarez et al. (Eds.): HAIS 2016, LNAI 9648, pp. 174–185, 2016.
DOI: 10.1007/978-3-319-32034-2_15

In this work, an efficient forecasting algorithm for big data is introduced. The proposed method is based on the well-known nearest neighbours techniques [3] in machine learning. This choice is due to the good results reported when applied to datasets of small or moderate size. The algorithm has been developed in the framework Apache Spark under the Scala programming language. The algorithm has been tested on real–world big datasets, namely energy consumption, collected from a sensor network located in several buildings of a public university.

The rest of the paper is structured as follows. Section 2 describes a review of the existing literature related to the nearest neighbours algorithms for time series forecasting and to the different approaches of the nearest neighbours for big data published in recent years. In Sect. 3 the proposed method based on nearest neighbours to forecast big data time series is presented. Section 4 presents the experimental results corresponding to the prediction of the energy consumption coming from a sensor network of building facilities. Section 5 closes the paper giving some final conclusions.

2 Related Work

Predicting the future has fascinated human beings since its early existence. Accurate predictions are essential in economical activities as remarkable forecasting errors in certain areas may incur large economic losses. Therefore, many of these efforts can be noticed in everyday events such as energy management, natural disasters, telecommunications, pollution, and so forth.

The methods for time series forecasting can be roughly classified as follows: classical Box and Jenkins-based methods such as ARMA, ARIMA, ARCH or GARCH [1] and data mining techniques (the reader is referred to [9] for a taxonomy of these techniques applied to energy time series forecasting). Namely, data mining techniques based on the k nearest neighbours (kNN) have been successfully applied, providing competitive results [7,8,14]. However, these methods cannot be applied when big time series have to be predicted due to the high computational cost of the kNN.

Consequently, several MapReduce-based approaches to address the kNN algorithm in big data scenarios have been recently proposed. The authors in [17] study parallel kNN joins in a MapReduce programming model that involves both the join and the NN search to produce the k nearest neighbours of each point in a new dataset from an original dataset. In particular, both exact (H-BRJ) and approximate (H-zkNNJ) algorithms are proposed to perform efficient parallel kNN joins on big data. In [11], an algorithm is proposed to address the problem of the fast nearest neighbour approximate search of binary features in high dimensional spaces using the message passing interface (MPI) specification. A MapReduce-based framework focused on several instance reduction methods is proposed in [13] to reduce the computational cost and storage requirements of the kNN classification algorithm.

In the context of this work, the kNN query is usually required in a wide range of sensor network applications. In fact, authors in [16] propose a MapReduce-based algorithm to generalize the spatio-textual kNN join in large-scale data.

This algorithm aims at searching text-similar and k-nearest sensors to a query set containing more than one query point.

Furthermore, other distributed architectures such as GPU have been used to address parallel versions of the kNN algorithm [2], also allowing a fast and scalable meta-feature generation for highly dimensional and sparse datasets.

Alternatively, the MLlib does not include any Spark implementation of the kNN algorithm, in spite of providing several traditional algorithms such as k-means, decision tree, among others. Thus, several parallel implementations of the kNN algorithm have been proposed in the literature. For instance, the work presented in [12] provides a Spark implementation in Java of the kNN and the SVM-Pegasos algorithms to compare the scalability of this parallelization technology with the MPI/OpenMP on a Beowulf cluster architecture. This kNN Spark implementation maps the Euclidean distance from each training sample to a test sample.

3 Methodology

This section describes the methodology proposed in order to forecast big data time series. In particular, Sect. 3.1 introduces the methodology itself and in Sect. 3.2 how it is implemented to be used in Spark.

3.1 Time Series Forecasting Based on Nearest Neighbours

This section describes the technique applied to time series forecasting based on the kNN algorithm.

Given the electricity consumption recorded in the past, up to c_i, the problem consists in predicting the h consecutive measures for electricity consumption (note that h is the prediction horizon).

Let $C_i \in \mathbb{R}^h$ be a vector composed of the h values to be predicted:

$$C_i = [c_{i+1}, c_{i+2}, \ldots, c_{i+h}] \tag{1}$$

Then, the associated vector $CC_i \in \mathbb{R}^w$ is defined by gathering the consumption contained in a window composed of w consecutive samples, from values of the vector C_i backwards, as follows:

$$CC_i = [c_{i-w+1}, c_{i-w+2}, \ldots, c_{i-1}, c_i] \tag{2}$$

For any couple of vectors, CC_i and CC_j, a distance can be defined as:

$$\text{dist}(i, j) = \|CC_i - CC_j\| \tag{3}$$

where $\|\cdot\|$ represents a suitable vector norm (the Euclidean norm has been used in this work).

The weighted nearest neighbours (WNN) method first identifies the k nearest neighbours of vector CC_i, where k is a number to be determined and *neighbourhood* in this context is measured according to (3) as afore mentioned. This leads to the neighbour set, NS:

$$NS = \left\{\text{set of } k \text{ indexes, } q_1, \ldots, q_k, \text{ such that } CC_{q_j} \text{ closest to vector } CC_i\right\} \tag{4}$$

in which q_1 and q_k refer to the first and k-th neighbours respectively, in order of distance.

According to the WNN methodology, the h electricity consumptions are predicted by linearly combining the consumptions of the k vectors succeeding those in NS, that is,

$$C_i = \frac{1}{\sum_{j=1}^{k} \alpha_j} \cdot \sum_{j=1}^{k} \alpha_j C_{q_j} \tag{5}$$

where the weighting factors α_j are obtained from,

$$\alpha_j = \frac{1}{(\text{dist}(CC_{q_j}, CC_i))^2} \tag{6}$$

Obviously, α_j when $j = k$ (furthest neighbour) is lesser than α_j when $j = 1$ (nearest neighbour). Note also that, although the w consumptions contained in CC_i are used to determine the nearest neighbours, only the h consumptions of the vectors C_{q_j} are relevant in determining C_i.

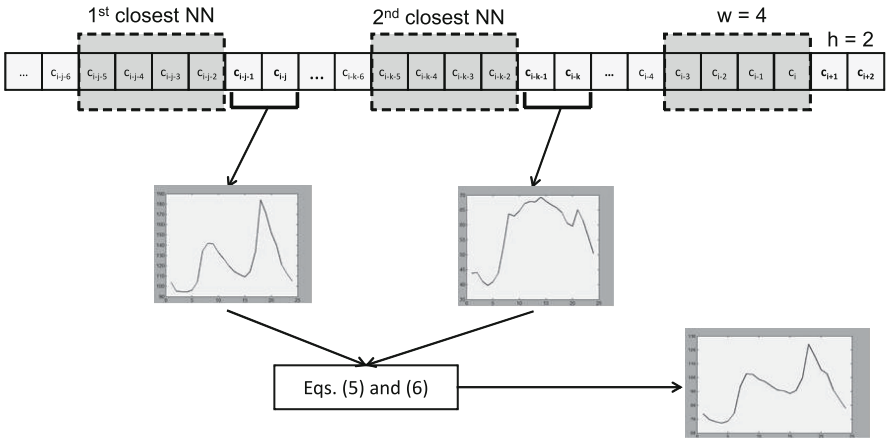


Fig. 1. Illustration of the WNN approach.

In order to find candidate neighbours, q_j, a window of w samples is simply slid along the entire dataset.

Figure 1 illustrates the basic idea behind the WNN algorithm, with $k = 2$ and $w = 4$. Values c_{i+1} and c_{i+2} ($h = 2$) are the target prediction. As $w = 4$, values $[c_{i-3}, c_{i-2}, c_{i-1}, c_i]$ are chosen as window. Later, minimal distances calculated according to Eq. (3) are searched for in the historical data. Sequences of values $s_1 = [c_{i-j-5}, ..., c_{i-j-2}]$ and $s_2 = [c_{i-k-5}, \ldots, c_{i-k-2}]$ are identified as the two nearest neighbours. In particular, s_2 is closer to w than s_1, and would therefore be denoted as q_2 and q_1, respectively. Finally, the forecast is performed by considering the h next samples to s_1 and s_2, according to Eqs. (5) and (6).

3.2 Algorithm Implementation for Apache Spark

The algorithm described in Sect. 3.1 has been implemented for Apache Spark, making the most of the RDD variables of Spark, in order to use it in a distributed way. This strategy makes the analysis of the datasets more efficient and faster. Therefore, every RDD created is split in blocks of the same size across the nodes that integrate the cluster, as it is shown in Fig. 2.

For a proper execution of the algorithm, several variables have to be defined from the beginning. These are:

1. The initial time series to be analysed.
2. The size of the window w whose values are taken as a pattern to look for the nearest neighbours.
3. The number of values h that needs to be predicted.
4. The number of nearest neighbours k that are going to be selected.

Since overwriting RDD variables cannot be done in Spark, a new RDD is created in each step of the algorithm. Hence, every time this section refers to a new transaction it means that a new RDD is being created.

Firstly, the data is loaded in Spark, split into different fields and finally just the energy consumption is selected, as shown in Fig. 3(a). An extra field with a numeric index is also added in this transaction. So the initial dataset in Spark is a RDD with just two fields, identification number with the position of the value of the time series, and the consumption itself (see Fig. 3(b)). Remark that, as

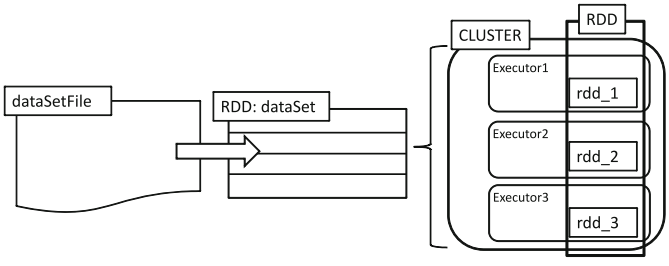
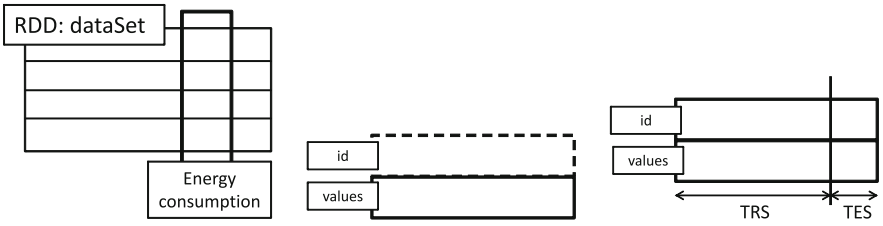


Fig. 2. Creation of a RDD variable in Spark and how it is managed in a cluster.

before mentioned, this data is split automatically across the nodes of the cluster. In a second transaction, the previous dataset is separated in two subsets, test set (TES) and training set (TRS), as Fig. 3(c) shows. TRS will be used to train the algorithm, whereas TES will be used to predict results and to check the accuracy of the prediction, comparing each predicted value to the actual one.



(a) Data selection. (b) Including an *id*. (c) TRS and TES.

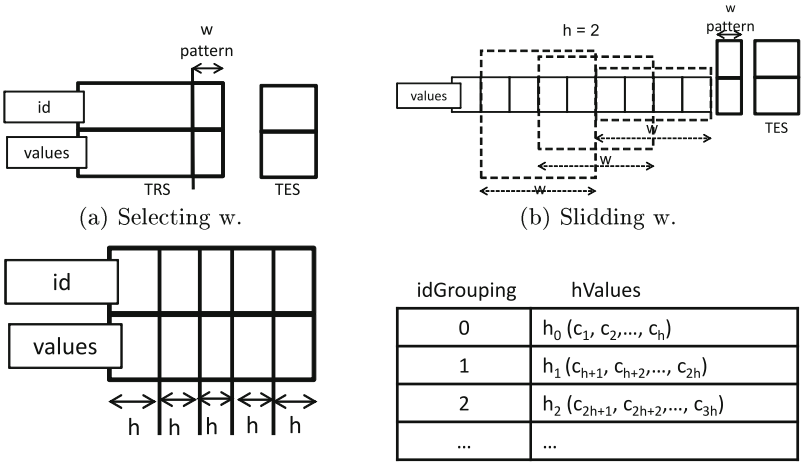
Fig. 3. Data preprocessing.

The next transaction only uses TRS for training purposes. Therefore, the w previous values to the h values to be predicted are selected as the pattern in this iteration, as depicted in Fig. 4(a). Now, the main goal is to store every possible subset of w values that can be formed of which their h future values are known. To achieve this, the windows w has to be shifted h values from the end of the time series, and from there, the w subsequent values are selected. Next iteration will repeat the same process as is illustrated in Fig. 4(b). For the following transaction, the TRS is divided in subsets of h values, as shown in Fig. 4(c). Thanks to map transformations that Spark provides to its RDD, this is done in one instruction all over the RDD located in the cluster. The key in this transaction is to group values just in one action without doing several iterations like it would have been done in other languages (Java 8's Stream is, perhaps, the sole exception). In this case, the RDD from the previous transaction is grouped by the rule id/h. As a result, the new RDD will contain a numeric id of the subsets following by their corresponding h values. This can be seen in Fig. 4(d), where *idGrouping* is the numeric id of each subset and h_i represents each subset of h values of the time series. In particular,

$$h_i = [c_{i \cdot h+1}, c_{i \cdot h+2}, ..., c_{(i+1) \cdot h}] \tag{7}$$

For instance, in the figure, h_0 is formed by the h values whose index id divided by h is 0, that is, $[c_1, c_2, ..., c_h]$.

As the formation of each subset of w values depends on the h_i previous subsets, a new RDD will be created with these subsets focusing on the RDD from the transaction before. So in this transaction, a dataset is also formed with the subsets of w values as well as the numeric id that matches with the RDD that contains the subsets h_i. This is represented in Fig. 5(a), where *idGrouping*



(a) Selecting w. (b) Slidding w.

(c) Not overlapped subsets of h values in TRS. (d) RDD with every possible h.

Fig. 4. kNN in Spark, phase 1.

is the numeric id of the subset and w_i represents each subset of w previous values to the h values of the subset h_i.

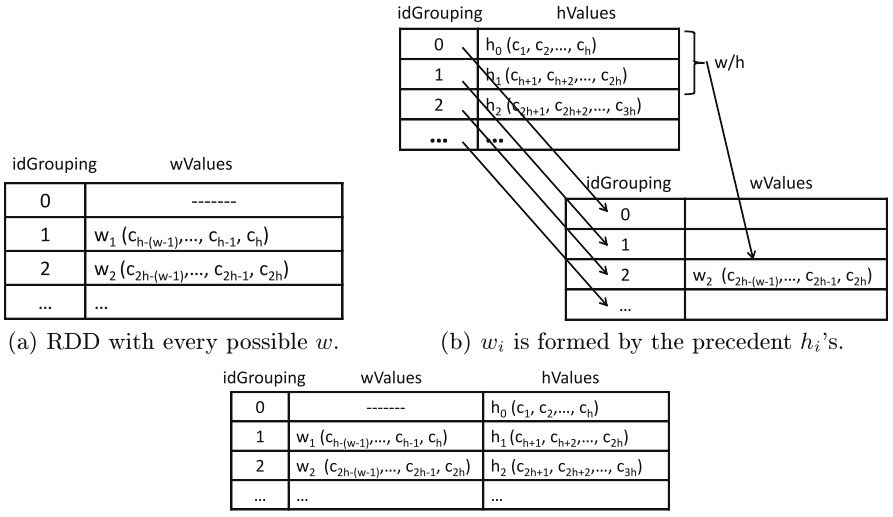
$$w_i = [c_{i \cdot h-(w-1)}, ..., c_{i \cdot h-1}, c_{i \cdot h}] \tag{8}$$

Due to the size of the windows w is usually greater than h, it can be observed that w_i cannot be defined when it does not exist w consecutive values previous to the values of h_i. For the same reason, it would be necessary not to look at just the values of the subset h_{i-1}, but in the w/h subsets before to build the subset w_i. This can be seen in Fig. 5(b), where the values for the w_2, for instance, are going to be formed by the values of the w/h previous h_i, in this case, h_0 and h_1.

In the next transformation, both RDDs need to be joined. Again, and thanks to the fact that both datasets share the same numeric id, Spark allows to do so by a simple action, obtaining a new RDD with the grouping id, the w values of the time series and the h values that follows it. This transaction can be seen in Fig. 5(c).

At this time the pattern w is compared to each w_i, obtaining the new field distance, which is calculated by the Euclidean distance and added to the previous RDD with just one action over Spark. Thus, the new dataset will contain the numeric id $idGrouping$, the w_i, the h_i and the distance d_i between the w pattern and w_i. This is shown in Fig. 6(a).

The next step of the algorithm sorts the previous RDD according to the distance, which Spark does rapidly just indicating the field for which the whole RDD is going to be sorted.



(a) RDD with every possible w.

(b) w_i is formed by the precedent h_i's.

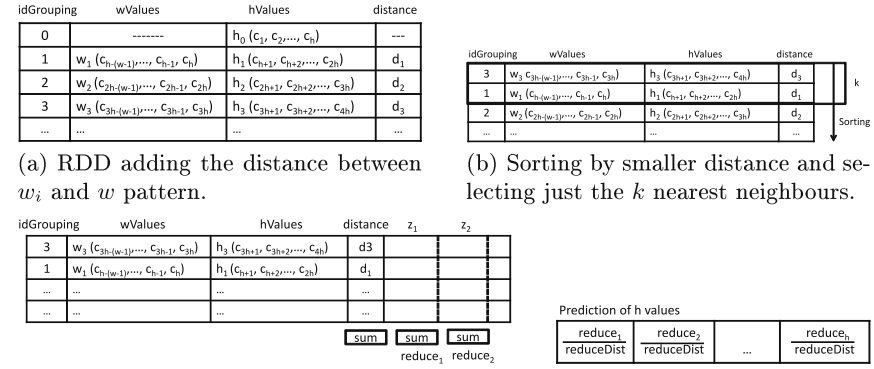
(c) RDD with joint w_i and h_i.

Fig. 5. kNN in Spark, phase 2.

After that, just the k nearest neighbours will be chosen. This is explained in Fig. 6(b), where k is the number of nearest neighbours to be selected.

The next transaction will calculate the prediction, applying the formulas (5) and (6). In Spark, before the value is predicted, it is necessary to do intermediate calculations. First, each value of each h_i is divided by the square distance. This is done in Spark with a new map transformation, adding a new field with the values of $h_i/(d_i)^2$. And finally, each of these fields needs to be summed, which in Spark is done with a reduce action over the field obtaining a number. This is illustrated in detail in Fig. 6(c), where the new columns z_j represents the division of the j-th value of each h_i between its square distance $(d_i)^2$, sum represents the sum of each column and $reduce_j$ is the name of the variable that gather that number. So first column z_1 will be the division of each $c_{i \cdot h+1}$ of each h_i between the distance $(d_i)^2$, then it will be summed and saved in the variable $reduce_1$. Then, it is just necessary to divide each sum of each field with the sum of the inverse of the square distance (reduceDist variable), obtaining the h values predicted as shown in Fig. 6(d).

Once all the predictions for the h values are made, the process begins again to obtain the following h forecasts, but this time updating the TRS, as shown in Fig. 7(a), where TRS *old* represents the initial TRS; and TRS *new* the new one including the previous TRS and the h real values previously predicted. The algorithm will then stop when the total predictions have the same size as the TES. This can be seen in Fig. 7(b). The final step lies in comparing the prediction with the real values in TES applying the formula of the mean relative error, defined in Eq. (9), as shown in Fig. 7(c).



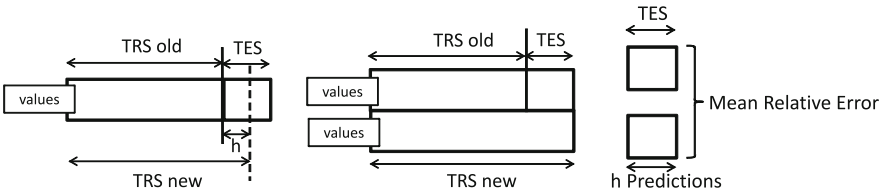
(a) RDD adding the distance between w_i and w pattern.

(b) Sorting by smaller distance and selecting just the k nearest neighbours.

(c) RDD adding suboperations as columns before doing reduce sums actions.

(d) Computation of the prediction.

Fig. 6. kNN in Spark, phase 3.



(a) TRS updating. (b) Algorithm finishing. (c) MRE calculation.

Fig. 7. kNN in Spark, phase 4.

4 Results

This section presents the results obtained from the application of the proposed methodology to electricity consumption big data time series. Hence, Sect. 4.1 describes the used datasets. The experimental setup carried out is detailed in Sect. 4.2. Finally, the results are discussed in Sect. 4.3.

4.1 Datasets Description

The datasets used are related to the electrical energy consumption in two buildings located at a public university for years 2011, 2012 and 2013. The consumption is measured every fifteen minutes during this period. This makes a total of 35040 instances for years 2011 and 2013, and 35136 for the year 2012. TES consists of the the last three months of 2013 for both datasets.

Note that there were several missing values ($< 3\,\%$). In particular, some values are equal to 0. However, subsequent time stamps store the accumulated consumption for such instances. Therefore, the cleaning process consisted in searching for such 0 values and assuming that consumption had been constant

during these periods of time. That is, the stored value after zeros is divided by the number of consecutive registered zeros and assigned to each one.

4.2 Design of Experiments

The proposed algorithm requires several variables to be executed. Since this is a preliminary study, an exhaustive analysis of best values has not been carried out and the following considerations have been taken into account:

1. The size of the window (w) has been set to 96, which represents all the values for a whole day.
2. As for the forecast horizon (h), it was firstly set to $h = 48$ (12 h) and secondly to $h = 96$ (24 h).
3. The number of nearest neighbours (k) varies from one to five.

The algorithm has been executed using each dataset described in Sect. 4.1 varying the aforementioned parameter settings (w, h and k). In short, each dataset has been executed 10 times.

To evaluate the runtime costs of the algorithm, each complete experimentation for each dataset has been executed using two processors, summing 20 executions in total.

The experimentation has been launched on a cluster, which consists of 2 Intel Xeon E7-4820 processors at 2 GHz, 18 MB cache, 8 cores per processor and 64 GB of main memory working under Linux Ubuntu. The cluster works with Apache Spark 1.4.1 and Hadoop 2.3.

Finally, in order to assess the performance of the algorithm, the well-known mean relative error (MRE) measure has been selected. Its formula is:

$$MRE = \frac{1}{N} \sum_{i=1}^{N} \frac{|v_{pred} - v_{actual}|}{v_{actual}} \tag{9}$$

where v_{pred} stands for the predicted values and v_{acutal} for the actual consumption values.

4.3 Electricity Consumption Big Data Time Series Forecasting

This section shows the results of applying the methodology proposed in Sect. 3.1 to the datasets described in Sect. 4.1 over the cluster described in Sect. 4.2. The algorithm has been tested on the last three months in the year 2013, for both buildings, resulting in 8832 forecasted instances.

Table 1 summarizes the results obtained for the first building. Analogously, Table 2 shows the results obtained for the second building. Note that the column *Duration* collects execution times in minutes. The values for the rightmost columns show the MRE associated with each k.

It can be noticed that to facilitate future comparative analysis, only two processors have been used. Additionally, horizons of prediction has been set to

Table 1. Electricity consumption forecasting for the first building.

w	h	Duration	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
96	48	143.99	0.3184	0.2902	0.3025	0.3132	0.3090
96	96	53.00	0.4653	0.4243	0.4530	0.4773	0.4708

Table 2. Electricity consumption forecasting for the second building.

w	h	Duration	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
96	48	143.79	0.3052	0.2749	0.2870	0.2912	0.3030
96	96	53.13	0.4807	0.4159	0.4407	0.4452	0.4686

48 (12 h) and 96 samples (24 h), thus representing usual short-term forecasting horizons in electricity consumptions.

It can be seen in both tables that the execution time varies along with the size of the h, up to 66 %. This is because the higher the value of h is, the lesser number of distances has to be computed. It can also be noticed that small values of k do not make significant difference to the accuracy of the predictions. In fact it just changes from one k to another by 5 %. Farthest studies need to be carried out to select the optimal k.

It must be admitted that the execution time is expected to be improved in future versions. This is partly due to the fact that the calculation of every w_i is the only part of the algorithm which is not made in a parallelized way, but in an iterative way. Since previous h_i must be checked to compute w_i, in some cases, two subsets w_i and w_j may have values from the same h_i. This means that for every w_i, all the previous h_i's need to be individually checked, and just the ones after it are discarded. In short, a formula that creates every w_i from the original time series following a MapReduce schema have not been found so far. Obviously, future research will address this issue.

5 Conclusions

An algorithm to forecast big data time series has been proposed. In particular, the algorithm is based on the weighted nearest neighbours paradigm. This work describes how to design it in Spark. It also provides results for real-world time series, e.g. electricity consumption for several buildings at a public university. The implementation has been launched on a 2-processor cluster generating satisfactory results in terms of both MRE and execution time. Future work is directed in integrating the code in the Spark MLlib as well as in reducing its computational cost.

Acknowledgments. The authors would like to thank the Spanish Ministry of Economy and Competitiveness, Junta de Andalucía, Fundación Pública Andaluza Centro de Estudios Andaluces and Universidad Pablo de Olavide for the support under projects TIN2014-55894-C2-R, P12-TIC-1728, PRY153/14 and APPB813097, respectively.

References

1. Box, G., Jenkins, G.: Time Series Analysis: Forecasting and Control. John Wiley and Sons, Hoboken (2008)
2. Canuto, S., Gonçalves, M., Santos, W., Rosa, T., Martins, W.: An efficient and scalable metafeature-based document classification approach based on massively parallel computing. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 333–342 (2015)
3. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theor. **13**(1), 21–27 (1967)
4. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
5. Machine Learning Library (MLlib) for Spark (2015). http://spark.apache.org/docs/latest/mllib-guide.html
6. Hamstra, M., Karau, H., Zaharia, M., Knwinski, A., Wendell, P.: Learning Spark: Lightning-Fast Big Analytics. O' Really Media, Sebastopol (2015)
7. Martínez-Álvarez, F., Troncoso, A., Riquelme, J.C., Aguilar, J.S.: Discovery of motifs to forecast outlier occurrence in time series. Pattern Recogn. Lett. **32**, 1652–1665 (2011)
8. Martínez-Álvarez, F., Troncoso, A., Riquelme, J.C., Aguilar, J.S.: Energy time series forecasting based on pattern sequence similarity. IEEE Trans. Knowl. Data Eng. **23**, 1230–1243 (2011)
9. Martínez-Álvarez, F., Troncoso, A., Asencio-Cortés, G., Riquelme, J.: A survey on data mining techniques applied to electricity-related time series forecasting. Energies **8**(11), 12361 (2015)
10. Minelli, M., Chambers, M., Dhiraj, A.: Big Data, Big Analytics: Emerging Business Intelligence and Analytics Trends for Today's Businesses. John Wiley and Sons, Hoboken (2013)
11. Muja, M., Lowe, D.G.: Scalable nearest neighbor algorithms for high dimensional data. IEEE Trans. Pattern Anal. Mach. Intell. **36**(11), 2227–2240 (2014)
12. Reyes-Ortiz, J.L., Oneto, L., Anguita, D.: Big data analytics in the cloud: spark on hadoop vs MPI/OpenMP on beowulf. Procedia Comput. Sci. **53**, 121–130 (2015)
13. Triguero, I., Peralta, D., Bacardit, J., García, S., Herrera, F.: MRPR: a mapreduce solution for prototype reduction in big data classification. Neurocomputing **150**, 331–345 (2015)
14. Troncoso, A., Riquelme, J.C., Riquelme, J.M., Martínez, J.L., Gómez, A.: Electricity market price forecasting based on weighted nearest neighbours techniques. IEEE Trans. Power Syst. **22**(3), 1294–1301 (2007)
15. White, T.: Hadoop, The Definitive Guide. O' Really Media, Sebastopol (2012)
16. Yang, M., Zheng, L., Lu, Y., Guo, M., Li, J.: Cloud-assisted spatio-textual k nearest neighbor joins in sensor networks. In: Proceedings of the Industrial Networks and Intelligent Systems, pp. 12–17 (2015)
17. Zhang, C., Li, F., Jestes, J.: Efficient parallel kNN joins for large data in mapreduce. In: Proceedings of the International Conference on Extending Database Technology, pp. 38–49 (2012)

6.2. Big data time series forecasting based on nearest neighbours distributed computing with Spark

El artículo publicado en revista indexada es el siguiente:

- R. L. Talavera-Llames, R. Pérez-Chacón, A. Troncoso, and F. Martínez-Álvarez. Big data time series forecasting based on nearest neighbours distributed computing with Spark. *Knowledge-Based Systems*, 161:12-25, 2018.
    - Estado: publicado (disponible on-line)
    - Índice de impacto (JCR 2017): 4.396
    - Área de conocimiento:
        - Computer Science, Artificial Intelligence. Ranking 14 / 132 - Q1

Knowledge-Based Systems 161 (2018) 12–25

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Big data time series forecasting based on nearest neighbours distributed computing with Spark

R. Talavera-Llames, R. Pérez-Chacón, A. Troncoso*, F. Martínez-Álvarez

Division of Computer Science, Universidad Pablo de Olavide, Seville ES-41013, Spain

ARTICLE INFO

Keywords:
Big data
Spark
Time series forecasting

ABSTRACT

A new approach for big data forecasting based on the k-weighted nearest neighbours algorithm is introduced in this work. Such an algorithm has been developed for distributed computing under the Apache Spark framework. Every phase of the algorithm is explained in this work, along with how the optimal values of the input parameters required for the algorithm are obtained. In order to test the developed algorithm, a Spanish energy consumption big data time series has been used. The accuracy of the prediction has been assessed showing remarkable results. Additionally, the optimal configuration of a Spark cluster has been discussed. Finally, a scalability analysis of the algorithm has been conducted leading to the conclusion that the proposed algorithm is highly suitable for big data environments.

1. Introduction

Sometimes, large quantities of data can be difficult to understand or even to analyse. Data mining techniques offer a solution to deal with this problem and gain knowledge from these data. As a matter of fact, data mining has gone one step even further, and has been used to forecast. Trying to predict earthquakes [12], breast cancer [24], urban traffic congestion [2] or even bivalve landings to help fishery authorities [29] are just some examples.

One of the fields that is attracting considerable attention is energy consumption forecasting. Over the last few years, governments and companies have focused on this topic as accurate predictions could lead to saving a large amount of money and have a better impact on the environment. Therefore, collecting as much data as possible and simultaneously studying the data already gathered is fundamental. Not only is the whole house being measured and recorded, but also every appliance within it as works [21,26] show. Moreover, other studies have gone one step further and have focused on reducing energy costs with cloud computing optimisation. In [15] authors proposed a model, the dynamic energy-aware cloudlet-based mobile cloud computing model (DECM), to select in real time the cloud server a mobile user should connect to. The main improvement of the model is energy saving, allowing efficient communications between user devices and cloud servers. In [13] a novel model to optimise task assignments in mobile computing to reduce energy costs called Energy-Aware Heterogeneous Cloud Management (EA-HCM) model is proposed. The authors achieved this reduction by using heterogeneous mobile embedded system (MES) in cloud computing generating near optimal solutions. In [14] the authors focused on the workload scheduling issue in Cyber-Physical Systems (CPS). They proposed a model, the Smart Cloud-based Optimising Workload (SCOW) model, to provide sustainable services using heterogeneous cloud computing. The authors developed several algorithms to create near-optimal solutions to allocate computing resources in heterogeneous clouds.

Nowadays, thanks to recent advances in hardware development, gathering information is no longer a problem. This had led to thousands of petabytes being collected every day. In this context, data mining techniques need to be revised in order to deal with such an amount of data. In light of these events, new frameworks to successfully implement data mining techniques have emerged. One of the most well-known is the MapReduce paradigm [9] developed by Google, which allows programmers to parallelise the computation of an algorithm across a cluster of machines, resulting in faster and more efficient data processing. The Apache Hadoop software [17] rapidly gained popularity amongst the science community as it successfully implemented the MapReduce paradigm as well as being open source software. However, Hadoop has shown several issues when it comes to developing iterative algorithms and consequently other proposals have emerged. Among them, Apache Spark open source software [38] has been proved to be one of the most powerful. As all the computation is done in memory in Spark, the execution of a programme will be significantly faster than Hadoop (up to 100 times more than Hadoop as

* Corresponding author.
E-mail addresses: rltallla@upo.es (R. Talavera-Llames), rpercha@upo.es (R. Pérez-Chacón), atrolor@upo.es (A. Troncoso), fmaralv@upo.es (F. Martínez-Álvarez).

https://doi.org/10.1016/j.knosys.2018.07.026
Received 20 October 2017; Received in revised form 13 July 2018; Accepted 15 July 2018
Available online 17 July 2018

Spark's official web page states). Configuring Spark in a cluster can be a tough task, and therefore, some studies have actually focused on its automation as in [11].

In this paper we propose a complete and exact general-purpose forecasting algorithm based on k weighted nearest neighbours (kWNN), with application to energy demand time series in the Spanish electricity market. The algorithm has been developed in the Scala programming language using the Apache Spark framework. The main goal of Spark is to generalise MapReduce in order to support new applications within the same engine. Spark uses its own fault-tolerant variables to automatically work in a distributed way, the Resilient Distributed Dataset (RDD). Hence, once a dataset is loaded in an RDD variable, Spark distributes the dataset in different chunks in every node across a cluster. Each partition of the RDD is kept in memory, thus each node computes its part of the RDD. Any operation over an RDD is executed in parallel in each partition. The optimisation to in-memory processing makes Spark faster than Hadoop because disk input/output is minimised. Therefore, a user can focus on the development of his application and not in how and where distribute data, the complexities of the parallelism or the fault tolerance.

Nonetheless, a Spark cluster has many configurations and parameters [18], most of them established by default or automatically. Some of them could have an impact on the performance of the cluster, and therefore, on the execution time of the algorithm. In order to make the most of the Spark cluster, some basic configuration were analysed in this work. Namely, the number of cores that have been used in each machine of the cluster, the number of partitions in which the dataset was divided and the number of machines that have been used in the cluster were optimised to improve the performance of the proposed algorithm.

Longer times might be caused by two common problems: traffic analysis and management, and performance analysis of transmission control. Although this is not the main focus of the paper, there are some interesting works that pay attention to these issues [19].

The proposed algorithm starts by splitting the input dataset into the training set and the test set. Firstly, the training set is used to automatically select the optimal values of the input parameters. These are the w number of past values to be used as a pattern to look for its closest neighbours and the k number of closest neighbours. Some iterative steps still remain in the algorithm to forecast a whole test set, as the total prediction is calculated in intervals of h values, where h is the prediction horizon. Therefore, every time an interval is predicted, the corresponding real values are added to the training set. Finally, the test set is used to check the accuracy of the prediction. The high accuracy obtained demonstrates the suitability of the kWNN algorithm for distributed time series forecasting as it is shown in this work. The scalability of the algorithm has also been tested over two different clusters with different configurations such as number of cores per machine, number of machines in the cluster or the number of partitions of the dataset, highlighting the optimal ones.

In summary, the main contributions of this work are:

1. We propose a new approach based on the kWNN for big data time series forecasting.
2. Due to the high computational cost of finding k nearest neighbours, we develop the algorithm using an efficient distributed computing so that it can process very large time series.
3. We conduct a wide experimentation using real electricity data, measure every 10 min for ten years, from the Spanish electricity market. Reported results show the suitability of the algorithm with a 1.63% error.
4. We evaluate and compare the prediction accuracy of the proposed algorithm with five state-of-the-art big data forecasting approaches such as deep learning [35], decision tree, gradient-boosted tree, random forest and linear regression [16] improving the other comparing methods, on average, by 39.68%.
5. We carry out a scalability study with the purpose of obtaining the optimal design of the parameters related to the cluster configuration. Parameters such as the number of cores being used in a machine, number of partitions the dataset will be split into and number of nodes in a cluster have a great impact on the performance of the algorithm. We found that 16 slaves, 8 cores per node and 72 partitions are the optimal design for the time series under study.

The rest of the paper is structured as follows. Section 2 describes a review of the state-of-the-art approaches related to the nearest neighbours algorithm to time series forecasting problems. Section 3 presents the methodology used, namely, a comparison analysis of different weights, how the data is preprocessed, how in the training phase the optimal values of the parameters required for the algorithm are selected and how the prediction is made. Section 4 highlights the experiments carried out and the results obtained in terms of both accuracy and scalability of the algorithm in a cluster, along with the description of the dataset used. Finally, in Section 5 the conclusions drawn in this work are presented.

2. Related work

The k nearest neighbours algorithm (kNN) is a supervised method used for classification and it is considered as one of the most influential in data mining [37].

The kNN algorithm has already been used in the field of time series forecasting. In [4] the authors developed a new technique based on kernel regression using kNN for building energy modelling. Although they achieved good results in terms of prediction and performance for small training sets, they had to use a kNN approximation due to high computing costs.

Furthermore, a kNN algorithm is analysed, among other machine learning algorithms, and compared to a statistic-based algorithm in [21]. This work does not try to introduce a new algorithm to predict energy time series, but to compare some existing ones with a simple statistic algorithm in order to check which one offers both good performance and reasonable accuracy. The study demonstrates that the statistic-based algorithm achieves good accuracy results. In fact, the accuracy results are similar to the machine learning algorithms for energy use time series forecasting in a whole house and for appliance-level. The only issue is a decrement of computational costs.

On the other hand, a new algorithm, called Maximum Length Weighted Nearest Neighbour (MLWNN), is presented in [8]. The MLWNN is based on the WNN algorithm [36], which calculates the prediction depending on the distance to the nearest neighbours from a past window of values. This new algorithm is compared, in terms of performance, with other prediction algorithms such as WNN, an iterative neural network [31] and the PSF algorithm [27], which is a forecasting algorithm based on similarity of pattern sequences. For the experimentation phase, hourly electricity load time series collected during two years from three different countries are used. MLWNN obtained better or similar results as the other state-of-art methods proposed. No mention of its computational cost is reported.

In [32] a methodology based on kNN for long-term time series forecasting is proposed. They faced computational time issues by means of a feature selection strategy. Hence, kNN is used along with mutual information and nonparametric noise estimation to select the best feature set from the original dataset to forecast long-term time series, as they have been shown to be fast. A least squares support vector machines is used to predict a Polish electricity load time series over of 4 years during the 1990s.

Apart from kNN, many other techniques have been used to forecast time series. In [10] a review of techniques applied to forecast time series in the field of building energy consumption is presented. Short, medium and long term forecasting time series were considered in this study. Some of the techniques reviewed are ANN, ARIMA and SVM,

with a special focus on hybrid models, which combine several machine learning techniques achieving better results than the other techniques. On the other hand, some works have successfully applied distributed algorithms in Spark to predict energy consumption. One example is the study in [30] where a distributed k-means algorithm is used over a dataset of energy consumption in buildings of a public university. Another example is the study in [35] where deep learning is applied to electricity consumption in Spain. Another example is the study in [16] where several scalable methods, such as random forest and decision trees, are applied also to Spanish electricity load dataset.

In spite of the importance of the kNN algorithm, its computation usually requires a high resources cost due to the necessity to calculate every distance of the instances in the dataset to the pattern, and from them, choose the k closest ones. For that reason, the kNN can not be applied to big data and distributed programming and cloud computing technologies could offer a solution to this problem when big data time series need to be processed. Some works have already proved what cloud computing could offer. In [6] the author proposed a new service for financial institutions to improve risk analysis. These risks were based previously on the desktop and assuming a Gaussian distribution. As a result, a inaccurate and inadequate assessment of risk was carried out. The new service, Business Intelligence as a Service in the Cloud (BIaaS) make the most of cloud computing, adopting more complex models, improving accuracy in risk analysis, making predictions with small response times and with a low-cost investment. Another study [33] explores how to effectively deploy replica server in a content distribution network (CDN) in cloud data centres. The aim of the study is reducing response delays and bandwidth consumption by deploying sets of servers close to end users. Using virtual CDNs, authors constructed an adjacency graph and proposed a spectral clustering-based algorithm to solve it. A greedy algorithm was developed to complete up to 40 GB of data processing while considering the number of nodes. The experiments revealed high accuracy and reliability. The author in [7] proposes a system and application data science for weather cloud computing. Three goals were achieved in the study. First, the author develops a forecasting algorithm using a range of methods, choosing the optimal one to forecast each time a prediction needs to be made. The high accuracy achieved by the algorithm was demonstrated over data from three different parts of the world (Sydney, Singapore and London). Second, using five-step MapReduce, the author achieves short execution time in both data processing and analysis. Third, the author carries out data visualisation for temperature distributions using eight-step MapReduce. The data visualisation model was applied to two different locations in the world (the US and the UK) to show the suitability of the model.

In light of these advances, several works in order to reduce computing time when applying kNN for different tasks have been presented. In [25] a parallel kNN algorithm for classification tasks is provided. The experiments were carried out over several datasets of different sizes obtaining good results in terms of running time consumption.

On the other hand, in [23] a parallel approximate nearest neighbours algorithm is proposed in order to perform clustering over a repository of thousands of pictures.

For pattern recognition, authors in [34] proposed a facial recognition system for image tagging and classification. They used the kNN algorithm for the classification task in a Hadoop environment with a MapReduce processing.

Another field in which a scalable kNN has been used is in malware detection [5]. The aim of the authors in this work is to identify if a network node could be infected with malware. In order to do so, they first carry out a feature extraction from the proxy logs and then accomplish a classification, proposing a scalable kNN based on a MapReduce approach. Finally, in [1] two approaches using the MapReduce paradigm are presented in order to obtain kd-trees for large scale image retrieval in a distributed way. In this work, the kNN is executed in parallel, searching for and comparing every feature of an image to that of images stored in the dataset.

Conclusively, some surveys have been published collecting the latest works in which a distributed kNN has been developed, as seen in [28], where more than 50 studies are classified depending on a specific taxonomy such as the application of the study or the dimensionality of the dataset used. However, to the authors' knowledge, none of them were developed to forecast very large time series. In summary, the study of the related work reveals that the kNN algorithm has already being used for time series forecasting. However, just small datasets have been considered, facing high computational costs. In some works, even an approximate kNN algorithm had to be developed to reduce such a cost. This is the first work that addresses a kNN algorithm for big data time series forecasting.

3. Methodology

This section presents the methodology used to implement a forecasting time series algorithm. In particular, Section 3.1 introduces the kWNN algorithm to predict time series along with a comparison analysis of different weights, Section 3.2 shows how the data need to be grouped so they can be analysed, in Section 3.3 the calculations to carry out the prediction are explained and in Section 3.4, before the prediction itself is calculated, the training phase is described, that is to say, how the optimal values of the parameters required for the algorithm are selected.

3.1. Time series forecasting based on nearest neighbours

The main purpose of the kWNN algorithm is explained as follows. Let us suppose a C^i vector with w values up to sample i that is, $C^i = [c_{i-w}, \dots, c_i]$.

In order to forecast the next h values, the kWNN searches for the k closest neighbours to C^i. Afterwards, a weight is calculated for each neighbour depending on the distance to the C^i vector, and the prediction is made by applying a weighted average of the h samples following those k closest neighbours. Although in the time series context the dynamic time warping algorithm to find patterns has been widely used with good results, as works such as [20] demonstrate, the euclidean distance has been the method used in this work to extract similar patterns. That is due to the forecasting algorithm used in this paper adapts the weighted nearest neighbours (WNN) [36]. The WNN look for similar patterns in the historical data depending on a concrete distance, and predicts with a weighted mean of the next value from the patterns found. Therefore, the proposed method includes the pattern behaviour in its own definition.

3.1.1. Analysis of different weights

Weights are crucial elements in the kWNN algorithm as it is the key in the prediction that will be made. Therefore, the weight selection was based on the assumption that the closer a neighbour is, the more influence it will have on the prediction. Thus, four weights were considered to be used in the algorithm. The definition of all of them are listed below:

$$\alpha_1 = \frac{1}{(dist_j)} \tag{1}$$

$$\alpha_2 = \frac{1}{(dist_j)^2} \tag{2}$$

$$\alpha_3 = \frac{1}{b + (dist_j)^2} \tag{3}$$

$$\alpha_4 = e^{-\frac{(dist_j)^2}{\sigma^2}} \tag{4}$$

In all four weights, $dist_j$ is the distance of the jth nearest neighbour to the pattern, σ is its associated standard deviation and b is an

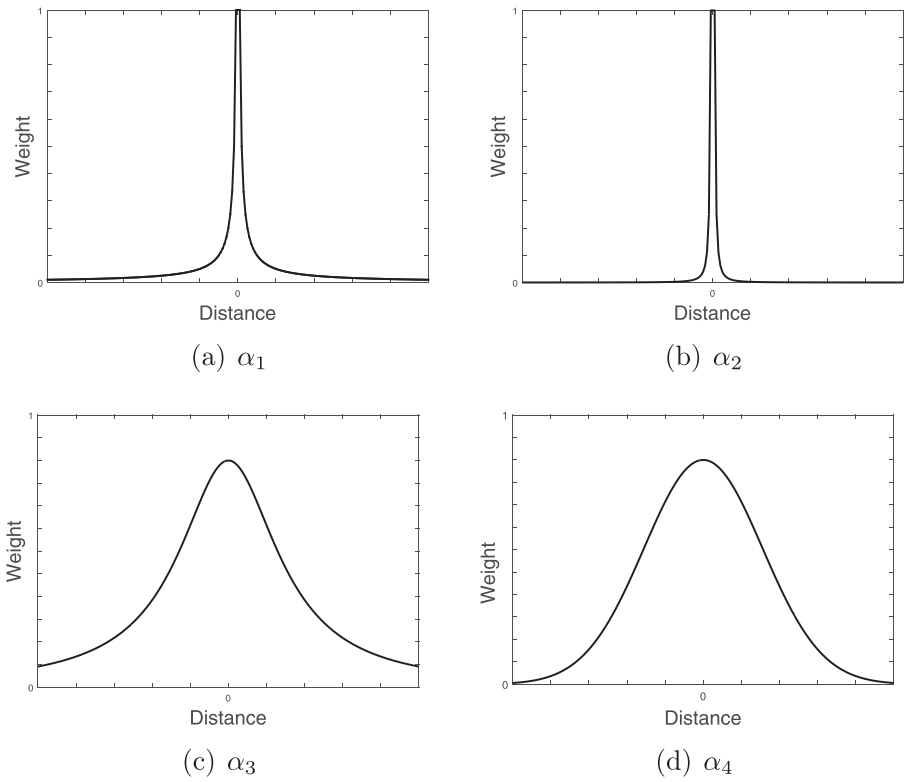


Fig. 1. Analysis of different weights.

independent term.

A behaviour study of the weights needs to be carried out. In order to do so, the weights were represented in the Fig. 1. As it can be seen, the distance-dependent function that gives a greater weight to the nearest neighbours corresponds to α_2, which is the selected weight.

3.1.2. The kWNN algorithm

The algorithm requires several variables in order to be executed from the beginning. These are:

1. The input time series to be analysed.
2. The maximum number of nearest neighbours that will be considered in order to automatically select just the *k* optimal ones.
3. The maximum size of a window of past values to automatically select just the *w* optimal ones.
4. The number of values to be predicted, *h*, in every iteration of the algorithm.

In order to select the optimal values of the parameters *w* and *k* to make a prediction, the algorithm first carries out two calculations: The first one, the false neighbour calculation, to obtain the optimal size of the window that will be formed to search for its closest neighbours. The second one, the optimal k calculation, that will result in the optimal number of nearest neighbours to select in order to calculate the prediction. Both operations are executed over a training set (TRS) as it can be seen in Fig. 2.

As w = 4, values $[c_{i-3}, c_{i-2}, c_{i-1}, c_i]$ represent the window of *w* past values to its next *h* values that the algorithm will predict. It will begin comparing this window to every window of *w* values that can be formed over the training set, and calculating the distance to them. Once done, it will select just the k closest neighbours, 2 in this case, as can be seen in Fig. 2. Then the next *h* values from both neighbours are selected, and applying a weighted average, the prediction is made. Finally, the accuracy of the prediction is measured comparing it with the real values in the test set (TES) from the input dataset. For further information, we recommend to check the state-of-art WNN approach in [36].

3.2. Preparing the data to be analysed.

Before the algorithm begins to calculate the nearest neighbours, it requires the time series to be organised in every possible window of *w* past values that can be formed with its next *h* known future values. To reach this goal, several operations are applied over the time series, as explained in this section.

As it can be seen in Fig. 3, there are no repeated values in the windows composed of the *h* future values but they are consecutive. In order to obtain all the possible groups of *h* values that can be formed from the original time series, we group each value of the time series c_j by the rule *id/h*, where *id* is its position. This is done in Spark in just one operation over the whole time series stored in a typical distributed variable of Spark, known as Resilient Distributed Dataset (RDD), as explained in Algorithm 1. For instance, in Fig. 3, c_0 and c_1 values will be past values to the *h* values with *idGrouping* 0, as 0/2 and 1/2 both are equal to 0.

However, it should be noted that to create the windows of *w* values there will be repetitions of the data in different windows as the formation of each one depends on shifting *h*, as the Fig. 3 shows. Namely, values of the time series c_2 and c_3 will be part of two windows as $w/h = 2$, that is to say to the windows with *idGrouping* 2 and 3.

Afterwards, *flatMap* and grouping operations are carried out in order to form each one of the possible windows of *w* values. In more detail, with a *flatMap* transformation we create a new RDD ungrouping the data with the identifier of the window or windows where the data belong. It will follow this form $< id, datum, idGrouping >$. Once we gather all the individual data, we group by the *idGrouping* variable, obtaining each one of the possible windows of *w* values that could be formed. Each one of these operations can be seen in Algorithm 1.

Finally, once all the possible windows of *w* values and *h* are formed, only a *join* function will be required, as both the RDD of *w* values and the RDD of *h* values share the same *idGrouping*.

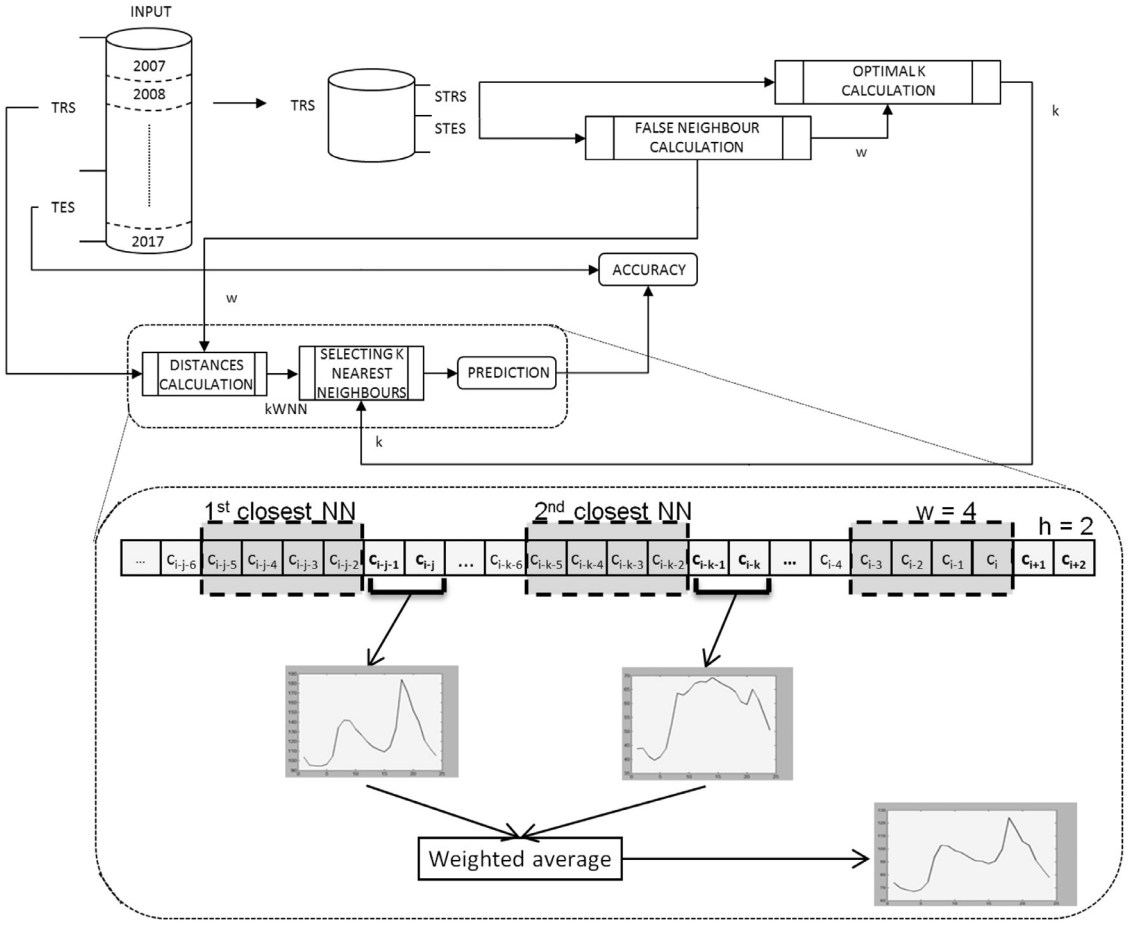


Fig. 2. Flowchart of the proposed kWNN algorithm.

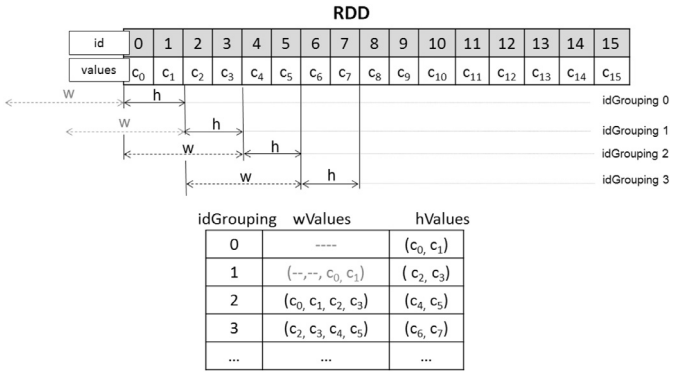


Fig. 3. A test example of windows of w past values and h future values.

3.3. Predicting

Once the optimal values for the size of the historical data window and the number of nearest neighbours have been computed, the kWNN algorithm finally makes a prediction from the RDD with every possible window of w past values and its next h values formed. This is shown in Fig. 4, where every RDD_i represents a variable across the cluster, meanwhile an rdd_{ij} represents a chunk of the RDD_i in the j executor of the cluster.

Fig. 5 presents the steps of the prediction, which are carried out in the cluster in a more detailed way. RDD_1 gathers the windows of w past values along with its next h values as seen in Fig. 5(a). This RDD needs to be split into training and test sets each time h values are predicted, as the TRS is updated adding these h real values from the TES as explained in Fig. 6. It needs to be mentioned that around 70% of the values will be the TRS and around 30% will be the TES.

In Fig. 5(b), a *filter* function is applied in order to select the TRS, being $n < m$. Afterwards, the w past values of the h values to predict are selected as the pattern to search for its k nearest neighbours, as shown in Fig. 6.

In the following transaction, the distance from the pattern to every window of w values is measured in a *map* function and added as a new column as shown in Fig. 5(c).

Next, the whole RDD is sorted in ascending order by the distances with a *sortBy* function and just the k with the closest distances to the pattern are selected with a *take* operation as shown in Fig. 5(d) and in Fig. 5(e), being k < n.

As these k registers represent a small number of values (between 1 and 20 instances in this work), all the operations from here are executed in the driver of the cluster, as shown in Fig. 4.

Subsequently, in order to calculate the prediction, each value from each window of h values is multiplied by an α_j weight calculated as showed in formula 2.

Next, the values in the same position are added together resulting in a vector of length h on the one hand, and the sum of the weights on the other hand, as seen in Fig. 7(a).

Finally, each of the values of the vector is divided by the sum of the weights, obtaining the final prediction, as shown in Fig. 7(b).

Then, the kWNN algorithm starts all over again, but adding h values from the TES to the TRS as h is the prediction horizon. When the size of the updated TRS reaches the total size of the original RDD, the algorithm will stop, and will return the accuracy of the prediction comparing to the TES.

Require: Dataset; w; h
1: Dataset form by RDD of (id, value)
2: **for** the whole RDD **do**
3: .groupBy(id/h)
4: **end for**
5: **Return** RDD_h
6: **for** the whole RDD **do**
7: .map(<id, $value$, ($Range$ [(id/h + 1) to (id/h + w/h)])>)
8: .flatMap(<id, $value$, $Range_w$>)
9: .groupBy($Range_w$)
10: **end for**
11: **Return** RDD_w
12: join(RDD_w, RDD_h)

Algorithm 1. Forming windows of w past values and its h future values.

3.4. Training the algorithm

Before the prediction is made, the optimal size of w and k is calculated. In order to do so, the next transactions only use the TRS for training purposes, specifically to choose the size of w with less than ten percent of false neighbours and the number of neighbours k with less prediction error for the STES validation set.

3.4.1. Selecting the size of window

In order to get the appropriate size of w, the false nearest neighbours method will need to be applied. This method will search for the number of false neighbours of a specific w window. It will start with the size of h, increasing it in the next stages of the method as $w = w + h$ until it reaches a given $maxW$ maximum value. When all possible values for w have been considered, the first one with less than ten percent of false neighbours will be chosen, as explained in Algorithm 2.

Having the original TRS, this is divided into another two subsets, the sub-training set (STRS) and the sub-test set (STES), as seen in Fig. 8(a). The STRS will be used to search for the closest neighbour of a window of w past values and will check if it is a false neighbour or not. The last w values of the STRS are selected as the pattern to compare to, gathering also its next h future values. In other words, the first h values of the STES, as shown in Fig. 8(b).

Before computing the number of false neighbours, it is necessary to form windows of w values with its next h future values from the input dataset. This is done as explained in Section 3.2, but using the STRS instead of the TRS.

The algorithm will continue in a similar way as explained in Section 3.3, comparing the w pattern to each window of w values formed, w_i, and obtaining the distance to it. But instead of calculating a prediction, the next h values from the pattern are compared to each next h values from each window of w values, h_i, obtaining a new field distance. This can be seen in Fig. 9(a), where the new RDD will contain the numeric identifier *idGrouping*, the w_i windows, the h_i windows, the distance dW_i between the w pattern and w_i and the distance dH_i between the h next values from the pattern and h_i.

The next transaction sorts the previous RDD in ascending order by the distance to the w pattern, dW, as seen in Fig. 9(b). After, just the nearest neighbour to the w pattern is chosen as shown in Fig. 9(c).

Finally, dH_i is divided by its associated dW_i corresponding to the nearest neighbour, and if the result is greater than one, this nearest neighbour is considered a false neighbour.

Next, h values from the STES are added to the previous STRS, in the same way as explained in Section 3.3. Once the size of the STRS is the same as the TRS, the function finishes and returns the total number of false neighbours found when predicting the STES for this particular w. Then, the falseNeighbour function is launched again, but with a different w size ($w = w + h$). To conclude, just the first w size whose number of false neighbours is smaller than ten percent of the total number of registers in STES will be chosen.

3.4.2. Optimal number of nearest neighbours

Once the optimal value for the size of the window has been fixed, the next step of the algorithm consists of choosing the optimal value of the nearest neighbours, k, whose relative prediction error will be the minimum error. Therefore, the main goal of this algorithm is to predict STES, based just on the STRS, and to measure the accuracy of this prediction for each k value from 1 to a specific maximum number. To achieve this, the TRS, the optimal w, the prediction horizon h and the maximum number of neighbours, $maxK$, to be considered will be required.

The algorithm will begin in the same way as explained in the false nearest neighbour algorithm. This means it will start splitting the *TRS* in *STRS* and *STES*. It will continue choosing the w pattern and forming every possible window of w values and its consecutive h values. Then, it will join the w and the h values in the same RDD and calculate the distance between the w pattern and every window of w past values. It will also sort this last RDD by distance in ascending order. However,

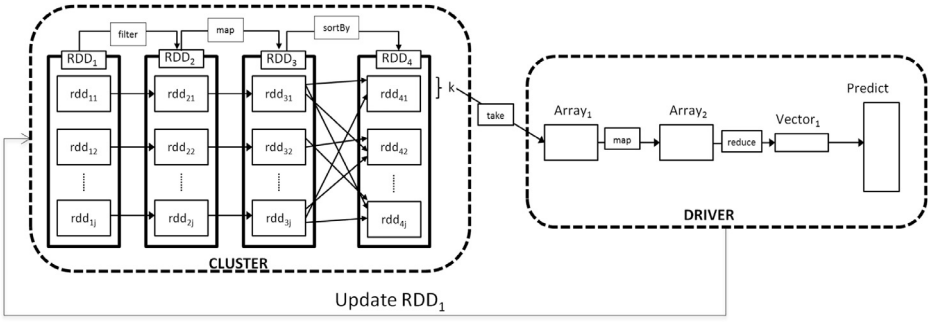


Fig. 4. Calculating the prediction.

idGrouping	wValues	hValues
0	----	$(c_{w-h}, c_{w-(h-1)},\ldots,c_{w-1})$
1	$(c_0,\ldots, c_{w-2}, c_{w-1})$	$(c_w, c_{w+1},\ldots,c_{w+(h-1)})$
2	$(c_h,\ldots, c_{w+h-2}, c_{w+h-1})$	$(c_{w+h}, c_{w+h+1},\ldots,c_{w+2h-1})$
...	...	...
m	$(c_{mh},\ldots, c_{w+mh-2}, c_{w+mh-1})$	$(c_{w+mh}, c_{w+mh+1},\ldots,c_{w+(m+1)h-1})$

(a) RDD_1 with every possible window of w and h values.

idGrouping	wValues	hValues
0	----	$(c_{w-h}, c_{w-(h-1)},\ldots, c_{w-1})$
1	$(c_0,\ldots, c_{w-2}, c_{w-1})$	$(c_w, c_{w+1},\ldots, c_{w+(h-1)})$
...	...	...
n	$(c_{nh},\ldots, c_{w+nh-2}, c_{w+nh-1})$	$(c_{w+nh}, c_{w+nh+1},\ldots, c_{w+(n+1)h-1})$

(b) RDD_2 with just the TRS training set.

idGrouping	wValues	hValues	dist
0	----	$(c_{w-h}, c_{w-(h-1)},\ldots, c_{w-1})$	---
1	$(c_0,\ldots, c_{w-2}, c_{w-1})$	$(c_w, c_{w+1},\ldots, c_{w+(h-1)})$	$dist_1$
...	...	...	...
n	$(c_{nh},\ldots, c_{w+nh-2}, c_{w+nh-1})$	$(c_{w+nh}, c_{w+nh+1},\ldots, c_{w+(n+1)h-1})$	$dist_n$

(c) RDD_3 with distances to pattern.

idGrouping	wValues	hValues	dist
7	$(c_{7h},\ldots, c_{w+7h-2}, c_{w+7h-1})$	$(c_{w+7h}, c_{w+7h+1},\ldots, c_{w+8h-1})$	$dist_7$
3	$(c_{3h},\ldots, c_{w+3h-2}, c_{w+3h-1})$	$(c_{w+3h}, c_{w+3h+1},\ldots, c_{w+4h-1})$	$dist_3$
...	...	...	...
i	$(c_{ih},\ldots, c_{w+ih-2}, c_{w+ih-1})$	$(c_{w+ih}, c_{w+ih+1},\ldots, c_{w+(i+1)h-1})$	$dist_i$

(d) RDD_4 sorted by distances.

idGrouping	wValues	hValues	dist
7	$(c_{7h},\ldots, c_{w+7h-2}, c_{w+7h-1})$	$(c_{w+7h}, c_{w+7h+1,}\ldots, c_{w+8h-1})$	$dist_7$
...	...	...	...
k	$(c_{kh},\ldots, c_{w+kh-2}, c_{w+kh-1})$	$(c_{w+kh}, c_{w+kh+1},\ldots, c_{w+(k+1)h-1})$	$dist_k$

(e) $Array_1$ selecting just the k closest neighbours.

Fig. 5. Steps of the kWNN algorithm carried out in the cluster.

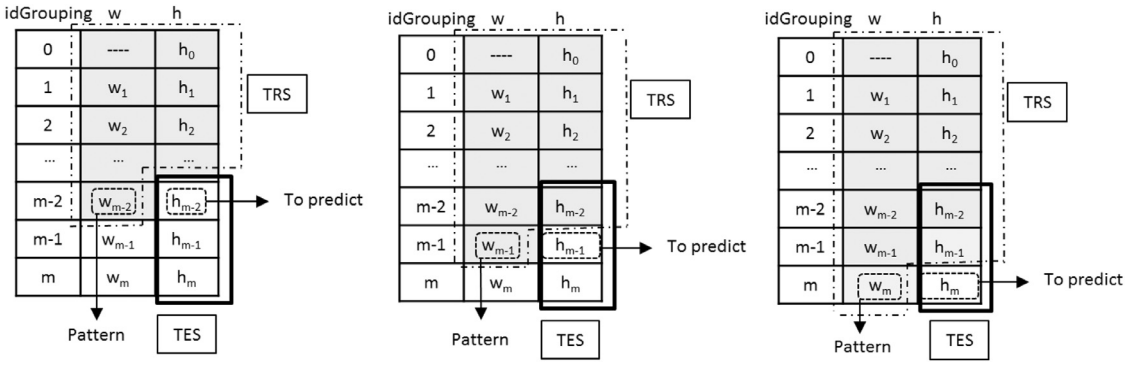


Fig. 6. Updating the TRS when h values are predicted.

this time, instead of selecting just the closest neighbour, as in the false nearest neighbour algorithm, it will choose the k closest neighbours. Finally, it will calculate the prediction for a particular k value.

Next, the following h values are predicted but updating the STRS with the h real values, predicted just now, from the STES to the STRS. Once the STRS reaches the total size of the TRS, the accuracy of the prediction of the STES for that k value is measured in the same way as explained in Section 3.3. Then, the relative error is computed again for the next k value as seen in Algorithm 3. The final step lies in choosing the k with the smallest relative error, when the relative error of the prediction over STES for each k has been measured.

4. Experiments and results

This section describes the experiments carried out and the results obtained after applying the kWNN algorithm proposed in the methodology to the electricity demand in Spain. In particular, Section 4.1 explains the dataset used. Section 4.2 shows the formulas of the metrics used to assess the performance of the method. Section 4.3 offers full details about the training of the algorithm. In Section 4.4 the results of the algorithm are discussed. Finally, the study of its scalability in order to get the best performance of the algorithm is presented in Section 4.5.

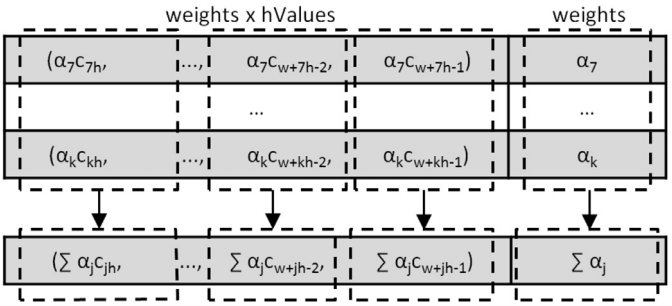
4.1. Dataset description

The dataset used is related to the electrical energy demand in the Spanish electricity market from 1st January 2007 to 21st June 2016. The demand is measured every ten minutes and is expressed in megawatt. This makes a total of 497,832 samples.

In order to obtain the optimal values for the size of the window w and the number of nearest neighbours k for the algorithm, just a 70% of the dataset has been used as the TRS training set, in particular from 1st January 2007 to 19th August 2013. Once the optimal w and k have been selected, the TES test set from 20th August 2013 to the end of the time series, 21th June 2016, is used to evaluate the performance of the algorithm.

4.2. Quality parameters

The chosen error to measure the accuracy of the prediction has been the mean relative error (MRE). The MRE is described as follows:



(a) Calculating a weighted average.

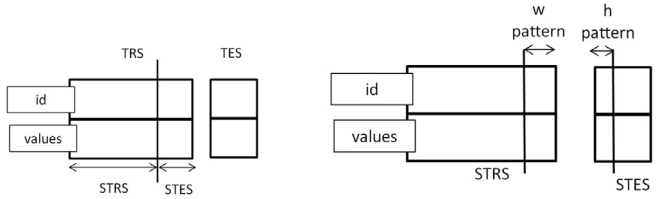
id	predict
0	$(1/\sum \alpha_j)\sum \alpha_j c_{jh}$
...	...
h-2	$(1/\sum \alpha_j)\sum \alpha_j c_{w+jh-2}$
h-1	$(1/\sum \alpha_j)\sum \alpha_j c_{w+jh-1}$

(b) Prediction.

Fig. 7. Steps of the kWNN algorithm carried out in the driver.

Require: TRS; maxW; h
1: TRS form by RDD of (id, value)
2: **for** $w \leftarrow h$ to $w < maxW$ **do**
3: $count[w]$=falseNeighbour(TRS, w)
4: $w = w + h$
5: **end for**
6: **Return** *count*

Algorithm 2. Computing false neighbours.



(a) Splitting the TRS dataset into STRS and STES.

(b) Choosing the w pattern and the h pattern to be compared.

Fig. 8. False neighbours in kWNN algorithm in Spark. Phase 1.

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\frac{|a_i - p_i|}{a_i} \quad (5)$$

where N represents the total number of predictions made, a_i represents one actual value and p_i represents the prediction for that actual value. Moreover, another error measure has been used in the experimentation of this work, the mean absolute error (MAE). The MAE express the exact amount of electricity above or below predictions, information of a significant importance for electricity companies in order to plan their energy production. The MAE is described as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|a_i - p_i| \quad (6)$$

As a study of the literature shows [3,22,36], MAE and MRE are the most accepted errors as they are considered to be the reference evaluation methods to measure the accuracy of the predictions in electricity demand time series forecasting.

idGrouping	wValues	hValues	dW	dH
0	-------	h_0 (c_{w-h}, $c_{w-(h-1)}$,..., c_{w-1})	---	---
1	w_1 (c_0,..., c_{w-2}, c_{w-1})	h_1 (c_w, c_{w+1},..., $c_{w+(h-1)}$)	dW_1	dH_1
2	w_2 (c_h,..., c_{w+h-2}, c_{w+h-1})	h_2 (c_{w+h}, c_{w+h+1},..., c_{w+2h-1})	dW_2	dH_2
3	w_3 (c_{2h},..., c_{w+2h-2}, c_{w+2h-1})	h_3 (c_{w+2h}, c_{w+2h+1},..., c_{w+3h+1})	dW_3	dH_3
...	...	...	...	

(a) RDD adding the distance between each w_i to w pattern and between each h_i to h pattern.

idGrouping	wValues	hValues	dW	dH
3	w_3 (c_{2h},..., c_{w+2h-2}, c_{w+2h-1})	h_3 (c_{w+2h}, c_{w+2h+1},..., c_{w+3h+1})	dW_3	dH_3
1	w_1 (c_0,..., c_{w-2}, c_{w-1})	h_1 (c_w, c_{w+1},..., $c_{w+(h-1)}$)	dW_1	dH_1
2	w_2 (c_h,..., c_{w+h-2}, c_{w+h-1})	h_2 (c_{w+h}, c_{w+h+1},..., c_{w+2h-1})	dW_2	dH_2
...	...	...	...	

Sorting by dW

(b) Sorting by smaller distance to the w pattern and selecting just the nearest neighbour.

idGrouping	wValues	hValues	dW	dH
3	w_3 (c_{2h},..., c_{w+2h-2}, c_{w+2h-1})	h_3 (c_{w+2h}, c_{w+2h+1},..., c_{w+3h+1})	dW_3	dH_3

(c) A simple vector with just the nearest neighbour.

Fig. 9. False neighbours in kWNN algorithm in Spark. Phase 2.

Require: TRS; optimal w; h; $maxK$;
1: TRS form by RDD of $(id, value)$
2: **for** $k \leftarrow 1$ to $k < maxK$ **do**
3: $count[k] = relativeError(TRS, w, h, k)$
4: **end for**
5: $k = argmin(count)$
6: **Return** k

Algorithm 3. Calculating the optimal number of neighbours.

4.3. Training phase

The experiments carried out to study and select the optimal w and k sizes are shown in this section. Several aspects need to be considered for a better understanding of the experiments:

1. Different values for the forecast horizon have been tested in order to analyse how sensitive the parameters of the algorithm w and k are. In particular:
   (a) $h = 24$, representing 4 hours.
   (b) $h = 48$, representing 8 hours.
   (c) $h = 72$, representing 12 hours.
   (d) $h = 144$, representing 24 hours.
2. Input variables required for the algorithm:
   - The maximum number of nearest neighbours, *maxK*, that are going to be analysed in order to select the k optimal one. This value has been set to 20.
   - The maximum size of the window, *maxW*, in order to select the w optimal one. This value has been set to ten times the size of h, that is, $10h$.

As explained in Section 3.4, firstly the TRS will be split into STRS and STES in order to train the algorithm. The results after executing this training phase will be shown in the following sections.

4.3.1. Selecting the optimal window size

As explained in Section 3.4.1, in order to obtain the optimal size of the window w, the false nearest neighbours algorithm is applied. The results of these analyses are shown in Table 1 and in Fig. 10. As it can be seen, for each prediction horizon, the first time the percentage of false nearest neighbours is lower than 10 percent is when w is 6 times h,

Table 1
Size of w selected for different prediction horizons.

	% False Neighbours			
#w size	#h = 24	#h = 48	#h = 72	#h = 144
h	0.8843	0.8140	0.7397	0.8532
2h	0.5236	0.4909	0.3699	0.4679
3h	0.3379	0.2652	0.2374	0.2661
4h	0.2344	0.1982	0.1689	0.2294
5h	0.1720	0.1341	0.1553	0.1193
6h	0.0807	0.0945	0.1142	0.0734
7h	0.0746	0.0823	0.1096	0.0459
8h	0.0578	0.0854	0.0731	0.0642
9h	0.0487	0.0671	0.0365	0.0550
10h	0.0381	0.0610	0.0320	0.0459

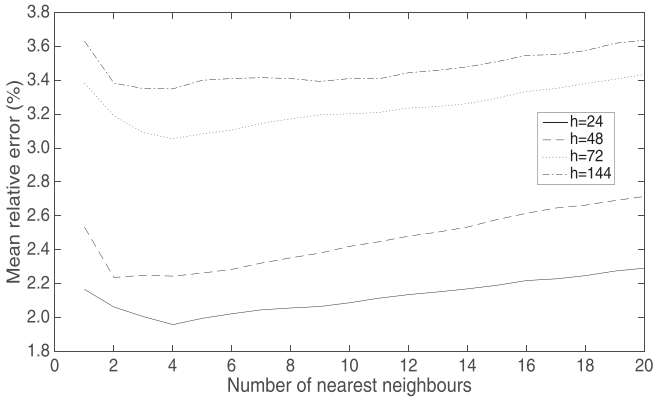
except when the size of h is 72, that is 8 times h. Which in summary leads to these optimal sizes of w:

- For h = 24, the optimal w size is 144, representing 24 hours (one day).
- For h = 48, the optimal w size is 288, representing 48 hours (two days).
- For h = 72, the optimal w size is 576, representing 96 hours (four days).
- For h = 144, the optimal w size is 864, representing 144 hours (six days).

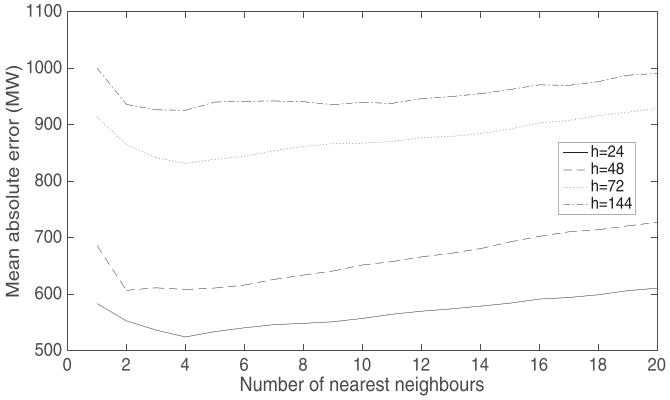
In short, it can be concluded that past values corresponding to one day are necessary to predict 4 hours. When h is 12 hours instead of three days one day more is required due to the existing noise in data.

4.3.2. Selecting the optimal number of nearest neighbours

Once the optimal w size has been set, the optimal number of nearest neighbours needs to be calculated. As discussed in Section 3.4.2, the STRS and the STES will be used, namely, for each prediction horizon, the prediction of the STES will be made from $k = 1$ to $k = 20$, selecting

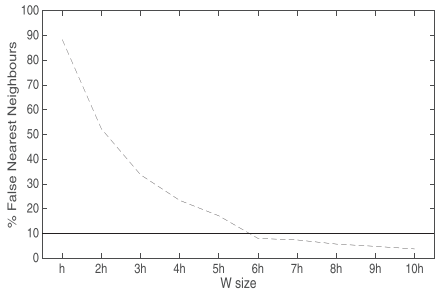


(a) Mean relative error.

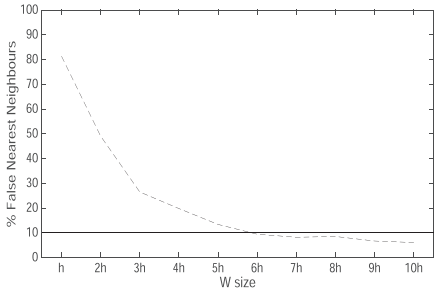


(b) Mean absolute error.

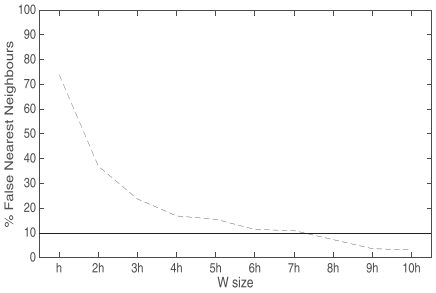
Fig. 11. Errors depending on the number of nearest neighbours for different prediction horizons.



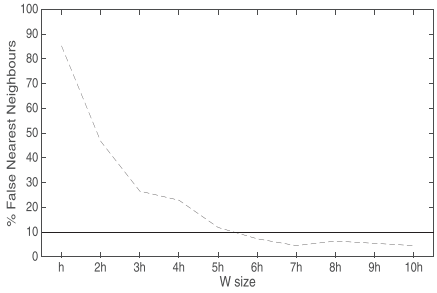
(a) h=24



(b) h=48



(c) h=72



(d) h=144

Fig. 10. Percentage of false nearest neighbours for different prediction horizons.

the number of nearest neighbours with the smallest MRE. Results are shown in Fig. 11. It should be noted that the optimal k number of nearest neighbours is 4 for every prediction horizon, except when the size of h is 48, meaning the optimal k is 2. In summary:

- For h = 24 and w = 144, the optimal k number of neighbours to be selected in order to obtain the best prediction is 4, corresponding to a mean absolute error (MAE) of 524.14 MW and a MRE of 1.90%.
- For h = 48 and w = 288, the optimal k number of neighbours to be selected in order to obtain the best prediction is 2, corresponding to a MAE of 920.87 MW and a MRE of 2.20%.
- For h = 72 and w = 576, the optimal k number of neighbours to be selected in order to obtain the best prediction is 4, corresponding to a MAE of 1313.40 MW and a MRE of 3.02%.
- For h = 144 and w = 864, the optimal k number of neighbours to be selected in order to obtain the best prediction is 4, corresponding to a MAE of 1514.92 MW and a MRE of 3.36%.

4.4. Results

Once the optimal w and k parameters have been set, the accuracy of the algorithm is computed, showing the results in this section. The prediction horizon h has been set to 24 (4 hours), in order to obtain the results provided here. Thus, the w window to be used as a pattern to look for its closest neighbours is set to 144 (24 hours), and the k number of closest neighbours to be selected to later calculate the prediction, is set to 4.

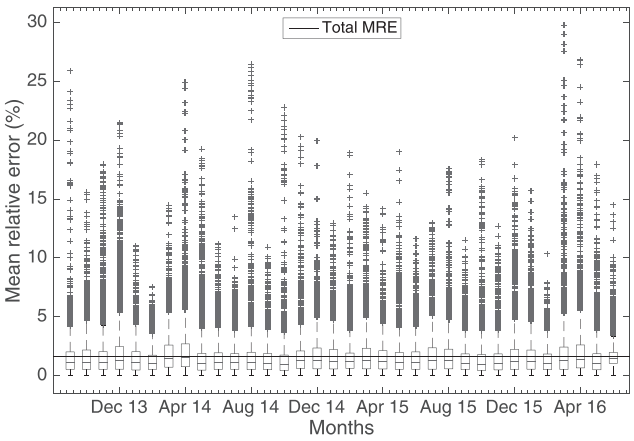
The MRE of the prediction for the test set has been 1.63%. In the following graphs, the results will be analysed.

Fig. 12 presents the mean relative errors of the prediction for each month. Although, it could be thought from Fig. 12(a) that there are many outliers errors, these do not represent a significant number due to the fact that 75% of all errors in every month are below 2.5%. As can be seen from Fig. 12(b), the highest error is 2.12% in April 2016, while the lowest MRE is 1.32% in February 2014. We will analyse February 2014 and April 2016 below.

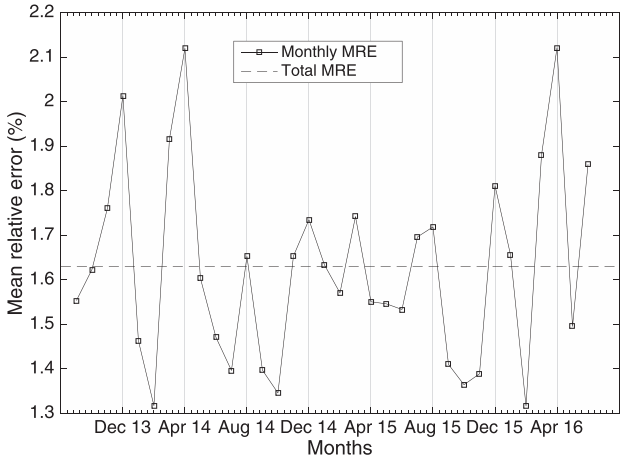
Fig. 13 represents the months with the lowest and highest MRE. For the best case representing February 2014, it can be seen that the MRE for each day remains steady around the MRE of the whole month. In fact, day 3 of the month achieves the worst MRE, with 2.47% meanwhile day 12 achieves the best MRE with 0.65%. On the other hand, for the worst case, April 2016, the MRE for each day varies significantly over the month achieving greater errors in days 1, 2 and 3 with 4.33%, 3.45% and 3.52%, respectively.

Fig. 14 shows the real energy consumption against those predicted for the best and the worst days in terms of mean relative error. Fig. 14(a) represents the day with the lowest mean relative error (0.45%), 17th July 2015. As can be seen, both the real and the predicted values are almost the same. On the other hand, Fig. 14(b) represents the day with the highest mean relative error (8.75%), 1st May 2014, a public holiday in Spain. It can be noted that the algorithm predicted values above the real ones but following the consumption pattern of the day. It starts with a decrease in the consumption, rising around 6:10am (value 37) when the real increase begins around 08:10am (value 49). However, the prediction falls again and rises around 07:10am (value 43), closer to the real value. Then, it continues with a similar line of consumption.

The MRE of the prediction for the whole test set has been 1.63%. Works [16,35], previously mention in Section 2, developed other big data methods using the same dataset in their predictions. To be more precise, [35] implemented a deep learning method meanwhile [16] compared predictions using methods based on trees, two ensembles techniques (gradient-boosted trees and random forests) and a linear regression. The accuracy of all these methods is compared in Table 2. The results of the state-of-the-art methods were accomplished with the settings described in [16,35], which are summarised below:



(a) MRE along with the standard deviation for the test set for each month.



(b) MRE of the prediction for the test set for each month.

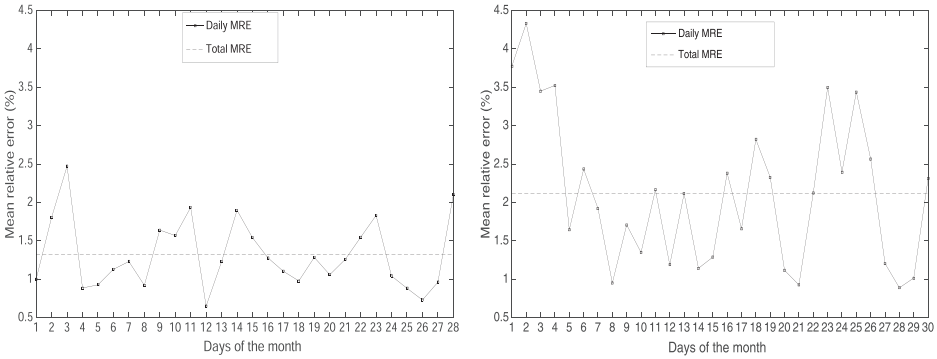
Fig. 12. Monthly relative prediction errors.

- Deep Learning: The activation function used was the hyperbolic tangent function. The distribution function chosen was the Poisson distribution. The lambda parameter to be used for regularisation of the dataset was set to 0.001. The Rho and the Epsilon parameters to describe the adaptive rate were set to 0.99 and 1.0E–9, respectively.
- Random forests: The number of trees was set to 50 and the depth was set to 8.
- Decision tree: The number of trees was set to 1 and the depth was set to 8.
- Gradient-Boosted trees: The number of trees was set to 5 and the depth was set to 8.
- Linear regression: The number of iterations and the rate of learning were set to 100 and 1.0E–10 as both parameters are required for the stochastic gradient descent method.

It should to be mentioned that the kWNN method obtained the best accuracy prediction among all methods; slightly better than the second best prediction, deep learning method, and 77.79% better than the worst prediction, linear regression method.

4.5. Scalability

Several experiments have been carried out with different configurations in two clusters in order to obtain the best performance of the algorithm. The results of these experiments are shown in this section, where the scalability of the algorithm is discussed.



(a) Month with the lowest MRE. (b) Month with the highest MRE.

Fig. 13. MRE for the best and worst month.

Focusing on the complexity of the algorithm, finding the closest neighbour of just one instance of the TES from a TRS would be $O(n \cdot D)$, being n the number of instances of the training and D the number of features. Therefore, finding the k nearest neighbours would have a greater computational cost as the distances calculated needs to be sorted. This adds another level of complexity resulting in $O(n \cdot \log(n))$. The process starts all over again for each instance of the test. Hence, the high computational cost of the kWNN makes impossible to use it when dealing with big data, thus justifying the need for developing a distributed kNN algorithm in order to obtain a scalable forecasting algorithm.

We first present the hardware and software used to analyse the scalability of the proposed method:

1. Cluster 1: It is composed of two nodes physically located at Data Science & Big Data laboratory, Pablo de Olavide University, with one master and one slave.
2. Cluster 2: It is a cluster in the cloud, composed of twenty-five nodes, one master and twenty-four slaves.

The nodes in every cluster are made up of the features show in Table 3.

The results obtained for the following experiments will be shown and discussed below.

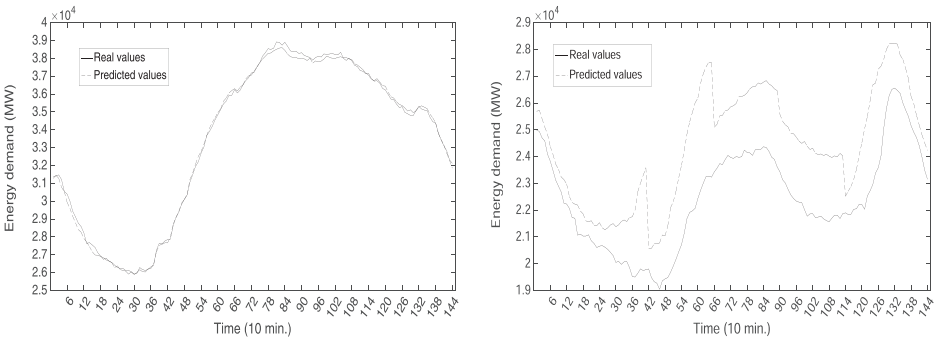
1. Speed up depending on number of cores. This experiment has been carried out over the cluster 1. The main goal was to discover how the algorithm responds when increasing the number of the cores of the machines. There are some details that need to be considered to better understand the results obtained:
   - The dataset is partitioned in the exact same number of pieces as the number of total cores the cluster is made up of. We guarantee with this configuration that every available core will be working with one of the chunks of the dataset. Hence, we make the most of Spark configuration, considering that it is advisable to partition the dataset in at least the same number of parts as cores that the cluster has.
   - For the 2 cores experimentation, the number of executors in the slave node was set to 1, while the RAM per executor was set to 15 GB.

Table 2
Accuracy comparison with other methods.

Method	MRE (%)
kWNN	1.63
Deep learning	1.84
Random forests	2.20
Decision tree	2.87
Gradient-Boosted trees	2.72
Linear regression	7.34

Table 3
Features of the nodes in the clusters.

	Cluster 1	Cluster 2
Hardware		
- Processors	Intel(R) Core(TM) i7-5820K CPU	Intel Xeon E5-2658 v3
- Cores	12 (12 threads)	8 (8 threads)
- Clock speed	2.13 GHz	2.2 GHz
- Cache	16 MB	30 MB
- RAM	16 GB	64 GB
Software		
- Framework	Apache Spark 1.6.1	Apache Spark 1.6.1
- OS	Ubuntu 16.04.1 LTS	Ubuntu 14.04.5 LTS



(a) Day with the lowest MRE. (b) Day with the highest MRE.

Fig. 14. MRE for the best and worst days.

Table 4
Speed up depending on number of cores.

#Total Cores	#Part	Time (hours)
2	2	4.8
4	4	3.6
6	6	2.95
8	8	2.57
10	10	2.3
12	12	2.11

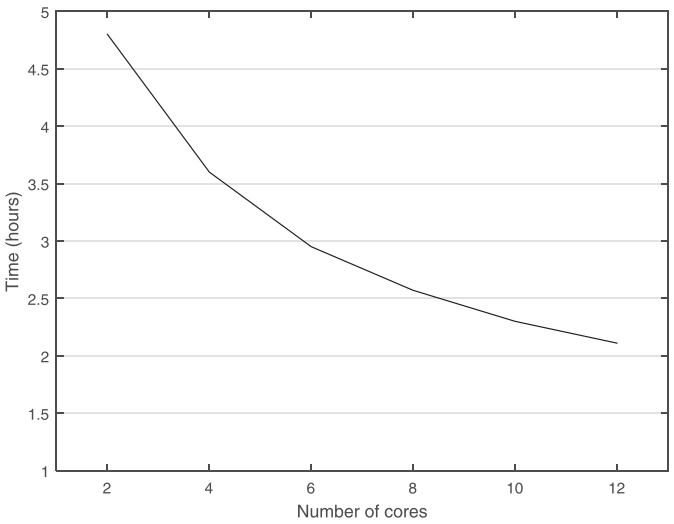


Fig. 15. Influence of number of cores on the algorithm.

- For the others, the number of executors in the slave node was set to 2, while the RAM per executor was set to 7 GB.
  Even though the 2 cores experimentation has greater RAM memory than the others, it takes the longest time to finish the execution of the algorithm, as can be seen in Table 4 and Fig. 15. On the other hand, the more cores the cluster has, the shorter the execution times will be. In particular, with 12 cores (and therefore with 12 partitions of the dataset) the time decreases by up to more than 50% compared to the 2 cores (and 2 partitions of the dataset) execution.

2. Speed up increasing the number of partitions for different cores. This experiment has also been carried out over the cluster 1. The main goal was to discover how the algorithm responds to increasing the number of partitions of the dataset in the machines for three different numbers of cores in each node. For each execution the slave node of the cluster has been set to 2 executors and 7 GB of RAM memory per executor.
   In summary, these are the different configurations:
   - Number of partitions of the dataset: 18, 36, 72 and 144.
   - Number of cores per executor: 4 (with a total number of 8 cores), 5 (with a total number of 10 cores) and 6 (with a total number of 12 cores).
     As can be seen in Table 5 and Fig. 16, all three configurations of the cores in the node of the cluster respond in the same way. In all three, the total time of the execution decreases while the number of partitions increases. However, from 72 partitions onwards, the execution time increases. Which leads us to the following conclusion. Better results are not always obtained with a greater number of partitions, in terms of speed. In addition, it should be mentioned that the 6-cores per executor configuration obtain the best results, as was expected. In particular, the total execution time for 72 partitions (1.5 hours) is 22% lower than that of 18 partitions, for example.

Table 5
Speed up depending on number of partitions of the dataset.

#Part.	Time (hours) 4 cores	Time (hours) 5 cores	Time (hours) 6 cores
18	2.07	1.96	1.93
36	1.78	1.72	1.6
72	1.73	1.58	1.5
144	2.02	1.85	1.77

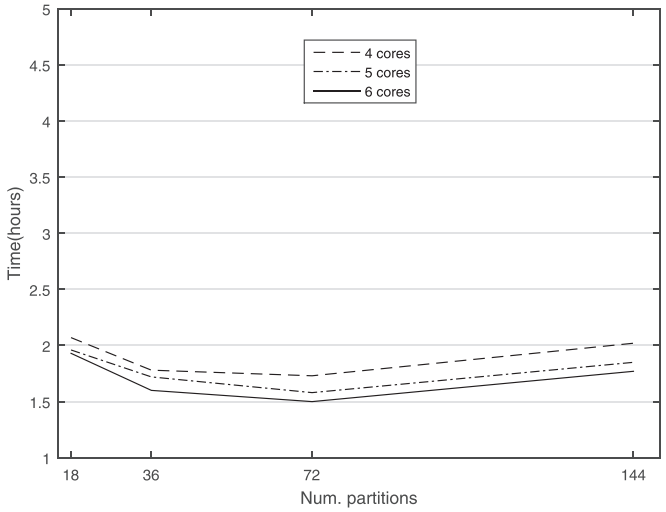


Fig. 16. Speed up depending on the number of partitions and the number of cores.

3. Speed up depending on the number of nodes in a cluster. This experiment has been carried out over the cluster 2. Its purpose was to analyse how the algorithm behaves with a different number of nodes in a cluster, once the number of partitions as well as the number of cores have been established following the results of the previous experiments.
   Several aspects need to be considered in advance in order to understand the results obtained in this experiment:
   - The number of partitions which the dataset is split into is set to 72 (according to the results obtained in the experiment *Speed up depending on number of partitions of the dataset*)
   - The number of total cores in each node of the cluster is set to the maximum number that every node has, which are 8 cores (following the results obtained in the *Speed up depending on number of cores* experiment)
   - The number of slaves considered for this experiment is 1, 2, 4, 8, 16 and finally 24 (the maximum number of nodes of the cluster)
   - Time consumption has been collected for two different numbers of executors in each node of the cluster:
     (a) One executor in each node with 8 cores and 60 GB RAM per executor.
     (b) Two executors in each node with 4 cores and 30 GB RAM memory per executor (which makes a total number of 8 cores and 60 GB RAM memory per node)
     From Table 6 and Fig. 17, the conclusion to be drawn is that

Table 6
Speed up depending on the number of slaves in a cluster.

#Slaves	Time (hours) with 1 Executor	Time (hours) with 2 Executor
1	2.05	2.13
2	1.85	1.58
4	1.5	1.2
8	1.13	1.08
16	1.07	1.07
24	1.11	1.12

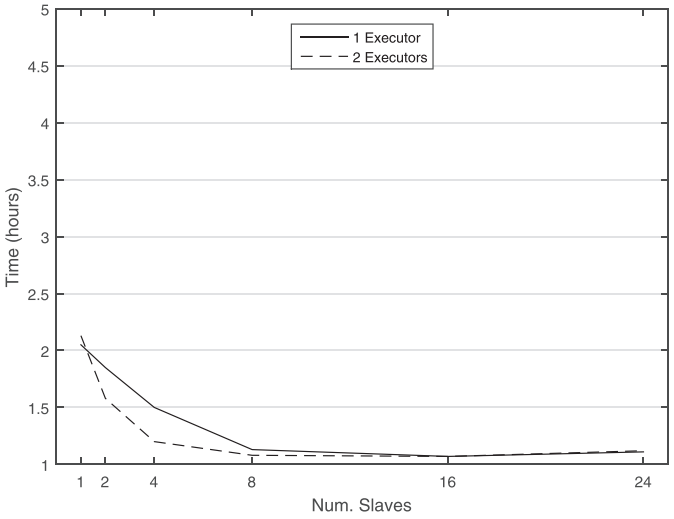


Fig. 17. Speed up depending on the number of slaves in a cluster.

with a higher number of slaves in the cluster, the time consumption decreases until 16 nodes, where the best results, in terms of execution time, are obtained. However, with 24 slaves in the cluster, time does not decrease more, it even partially rises in fact. It also needs to be mentioned that with a small number of nodes, such as 1, 2 or 4, configuring two executors per node results in better timing compared with 1 executor. Furthermore, with 8, 16 and 24 nodes the executions do not produce any significant difference. This leads us to establish that with a low number of nodes in a cluster it is advisable to set every node with two executors, meanwhile in a cluster with a great number of nodes there is no need to do so.

5. Conclusions

Due to the amount of data we are dealing with nowadays, data mining techniques need to be revised in order to offer good solutions with reasonable computational costs. In this paper, an exact and distributed algorithm to forecast big data time series has been presented: the k weighted nearest neighbours. The algorithm has been implemented over the Apache Spark framework in the Scala programming language, making the most of its in-memory executions. In order to correctly execute the algorithm, several parameters need to be defined, such as the w size of the windows and the k number of nearest neighbours to look for. The algorithm itself calculates the optimal values for these parameters before making the prediction. The results show the high accuracy of the prediction, demonstrating the suitability of the proposed algorithm for distributed time series forecasting. A dataset with more that 9 years of energy consumption in Spain, measured every 10 minutes, has been used to test the algorithm. A scalability study of the algorithm has also been carried out over two different clusters. The optimal configuration for the cluster, such as number of cores per machine, number of machines in the cluster or number of partitions of the dataset is also studied in this work, highlighting the optimal values. In future work, calculation of the distance to the closest neighbours will be improved using other methods, such as the dynamic time warping algorithm.

Acknowledgments

The authors would like to thank the Spanish Ministry of Economy and Competitiveness and Junta de Andalucía for the support under projects TIN2017-88209-C2-1-R, TIN2014-55894-C2-R and P12-TIC-1728, respectively. Additionally, the authors want to express their gratitude to the T-Systems Iberia company since all experiments were carried out on its Open Telekom Cloud Platform based on the OpenStack open source. Authors also thank Ms. Olivia Hope Rossall for her helpful comments.

References

[1] M. Aly, M. Munich, P. Perona, Distributed KD-trees for retrieval from very large image collections, Proceedings of the British Machine Vision Conference (BMVC), 17 (2011).
[2] G. Asencio-Cortés, E. Florido, A. Troncoso, F. Martínez-Álvarez, A novel methodology to predict urban traffic congestion with ensemble learning, Soft. Comput. 20 (11) (2016) 4205–4216, https://doi.org/10.1007/s00500-016-2288-6.
[3] A. Azadeh, S. Ghaderi, S. Sohrabkhani, Annual electricity consumption forecasting by neural network in high energy consuming industrial sectors, Energy Convers. Manage. 49 (8) (2008) 2272–2278, https://doi.org/10.1016/j.enconman.2008.01.035.
[4] M. Brown, C. Barrington-Leigh, Z. Brown, Kernel regression for real-time building energy analysis, J. Build. Perform. Simul. 5 (4) (2012) 263–276, https://doi.org/10.1080/19401493.2011.577539.
[5] P. Čech, J. Kohout, J. Lokoč, T. Komárek, J. Maroušek, T. Pevný, Feature extraction and malware detection on large https data using mapreduce, Proceedings of the Ninth International Conference on Similarity Search and Applications (SISAP), (2016), pp. 311–324, https://doi.org/10.1007/978-3-319-46759-7_24.
[6] V. Chang, The business intelligence as a service in the cloud, Future Gener. Comput. Syst. 37 (2014) 512–534. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications. 10.1016/j.future.2013.12.028 .
[7] V. Chang, Towards data analysis for weather cloud computing, Knowl. Based Syst. 127 (2017) 29–45, https://doi.org/10.1016/j.knosys.2017.03.003.
[8] T. Colombo, I. Koprinska, M. Panella, Maximum length weighted nearest neighbor approach for electricity load forecasting, Proceedings of the International Joint Conference on Neural Networks (IJCNN), (2015), pp. 1–8, https://doi.org/10.1109/IJCNN.2015.7280809.
[9] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113, https://doi.org/10.1145/1327452.1327492.
[10] C. Deb, F. Zhang, J. Yang, S.E. Lee, K.W. Shah, A review on time series forecasting techniques for building energy consumption, Renew. Sustain. Energy Rev. 74 (2017) 902–924, https://doi.org/10.1016/j.rser.2017.02.085.
[11] A.M. Fernández, J.F. Torres, A. Troncoso, F. Martínez-Álvarez, Automated spark clusters deployment for big data with standalone applications integration, Lect. Notes Artif. Intell. 9868 (2016) 150–159, https://doi.org/10.1007/978-3-319-44636-3_14.
[12] E. Florido, F. Martínez-Álvarez, A. Morales-Esteban, J. Reyes, J. Aznarte-Mellado, Detecting precursory patterns to enhance earthquake prediction in chile, Comput. Geosci. 76 (Supplement C) (2015) 112–120, https://doi.org/10.1016/j.cageo.2014.12.002.
[13] K. Gai, M. Qiu, H. Zhao, Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing, J. Parallel Distrib. Comput. 111 (2018) 126–135, https://doi.org/10.1016/j.jpdc.2017.08.001.
[14] K. Gai, M. Qiu, H. Zhao, X. Sun, Resource management in sustainable cyber-physical systems using heterogeneous cloud computing, IEEE Trans. Sustain. Comput. 3 (2) (2018) 60–72, https://doi.org/10.1109/TSUSC.2017.2723954.
[15] K. Gai, M. Qiu, H. Zhao, L. Tao, Z. Zong, Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing, J. Netw. Comput. Appl. 59 (2016) 46–54, https://doi.org/10.1016/j.jnca.2015.05.016.
[16] A. Galicia, J.F. Torres, F. Martínez-Álvarez, A. Troncoso, Scalable forecasting techniques applied to big electricity time series, Proceedings of the Fourteenth International Work-Conference on Artificial Neural Networks (IWANN), (2017), pp. 165–175, https://doi.org/10.1007/978-3-319-59147-6_15.
[17] S. Ghemawat, H. Gobioff, S. Leung, The Google file system, ACM SIGOPS Oper. Syst. Rev. 37 (5) (2003) 29–43, https://doi.org/10.1145/1165389.945450.
[18] A. Gounaris, J. Torres, A methodology for spark parameter tuning, Big Data Res. 11 (2018) 22–32.
[19] N. Hamid, V. Chang, R.J. Walters, G.B. Wills, A multi-core architecture for a hybrid information system, Comput. Electr. Eng. (2017), https://doi.org/10.1016/j.compeleceng.2017.12.020.
[20] S. Jeon, B. Hong, V. Chang, Pattern graph tracking-based stock price prediction using big data, Future Gener. Comput. Syst. 80 (2018) 171–187, https://doi.org/10.1016/j.future.2017.02.010.
[21] D. Lachut, N. Banerjee, S. Rollins, Predictability of energy use in homes, Proceedings of the International Green Computing Conference, (2014), pp. 1–10, https://doi.org/10.1109/IGCC.2014.7039146.
[22] H. Li, S. Guo, C. Li, J. Sun, A hybrid annual power load forecasting model based on generalized regression neural network with fruit fly optimization algorithm, Knowl. Based Syst. 37 (2013) 378–387, https://doi.org/10.1016/j.knosys.2012.08.015.
[23] T. Liu, C. Rosenberg, H.A. Rowley, Clustering billions of images with large scale nearest neighbor search, Proceedings of the IEEE Workshop on Applications of Computer Vision (WACV), (2007), pp. 28–33, https://doi.org/10.1109/WACV.2007.18.
[24] L. Macías-García, J.M. Luna-Romera, J. García-Gutiérrez, M. Martínez-Ballesteros,

J.C. Riquelme-Santos, R. González-Cámpora, A study of the suitability of auto-encoders for preprocessing data in breast cancer experimentation, J. Biomed. Inform. 72 (2017) 33–44, https://doi.org/10.1016/j.jbi.2017.06.020.
[25] J. Maillo, S. Ramírez, I. Triguero, F. Herrera, KNN-IS: an iterative spark-based design of the k-Nearest neighbors classifier for big data, Knowl. Based Syst. 117 (2017) 3–15, https://doi.org/10.1016/j.knosys.2016.06.012.
[26] F. Martínez-Álvarez, A. Troncoso, G. Asencio-Cortés, J.C. Riquelme, A survey on data mining techniques applied to electricity-related time series forecasting, Energies 8 (11) (2015) 13162–13193, https://doi.org/10.3390/en81112352.
[27] F. Martínez-Álvarez, A. Troncoso, J.C. Riquelme, J.S.A. Ruiz, Energy time series forecasting based on pattern sequence similarity, IEEE Trans. Knowl. Data Eng. 23 (8) (2011) 1230–1243, https://doi.org/10.1109/TKDE.2010.227.
[28] N. Nodarakis, A. Rapti, S. Sioutas, A.K. Tsakalidis, D. Tsolis, G. Tzimas, Y. Panagis, (A)kNN query processing on the cloud: a survey, Proceedings of the Second International Workshop on Algorithmic Aspects of Cloud Computing (ALGOCLOUD), (2017), pp. 26–40, https://doi.org/10.1007/978-3-319-57045-7_3.
[29] M.M. Oliveira, A.S. Camanho, J.B. Walden, V.L. Miguís, N.B. Ferreira, M.B. Gaspar, Forecasting bivalve landings with multiple regression and data mining techniques: the case of the portuguese artisanal dredge fleet, Mar. Policy 84 (2017) 110–118, https://doi.org/10.1016/j.marpol.2017.07.013.
[30] R. Pérez-Chacón, R.L. Talavera-Llames, F. Martínez-Álvarez, A. Troncoso, Finding electric energy consumption patterns in big time series data, Proceedings of the Thirteenth International Conference on Distributed Computing and Artificial Intelligence, (2016), pp. 231–238, https://doi.org/10.1007/978-3-319-40162-1_25.
[31] M. Rana, I. Koprinska, A. Troncoso, Forecasting hourly electricity load profile using neural networks, Proceedings of the International Joint Conference on Neural Networks (IJCNN), (2014), pp. 824–831, https://doi.org/10.1109/IJCNN.2014.6889489.
[32] A. Sorjamaa, J. Hao, N. Reyhani, Y. Ji, A. Lendasse, Methodology for long-term prediction of time series, Neurocomputing 70 (16) (2007) 2861–2869, https://doi.org/10.1016/j.neucom.2006.06.015.
[33] G. Sun, V. Chang, G. Yang, D. Liao, The cost-efficient deployment of replica servers in virtual content distribution networks for data fusion, Inf. Sci. (Ny) 432 (2018) 495–515, https://doi.org/10.1016/j.ins.2017.08.021.
[34] K. Sun, H. Kang, H. Park, Tagging and classifying facial images in cloud environments based on KNN using Mapreduce, Optik Inte. J. Light Electron. Optics. 126 (21) (2015) 3227–3233, https://doi.org/10.1016/j.ijleo.2015.07.080.
[35] J.F. Torres, A.M. Fernández, A. Troncoso, F. Martínez-Álvarez, Deep learning-based approach for time series forecasting with application to electricity load, Proceedings of the International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC), (2017), pp. 203–212, https://doi.org/10.1007/978-3-319-59773-7_21.
[36] A. Troncoso, J.M. Riquelme-Santos, A. Gómez-Expósito, J.L. Martínez-Ramos, J.C. Riquelme, Electricity market price forecasting based on weighted nearest neighbors techniques, IEEE Trans. Power Syst. 22 (3) (2007) 1294–1301, https://doi.org/10.1109/TPWRS.2007.901670.
[37] X. Wu, V. Kumar, J.R. Quinlan, J. Ghosh, O. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, Z. Zhou, M. Steinbach, D.J. Hand, D. Steinberg, Top 10 algorithms in data mining, Knowl. Inf. Syst. 14 (1) (2008) 1–37, https://doi.org/10.1007/s10115-007-0114-2.
[38] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, Proceedings of the Ninth USENIX Conference on Networked Systems Design and Implementation (NSDI), (2012), pp. 15–28.

6.3. MV-kWNN: A novel multivariate and multi-output weighted nearest neighbours algorithm for big data time series forecasting

El artículo publicado en revista indexada es el siguiente:

- R. L. Talavera-Llames, R. Pérez-Chacón, A. Troncoso, and F. Martínez-Álvarez. MV-kWNN: A novel multivariate and multi-output weighted nearest neighbours algorithm for big data time series forecasting. *Neurocomputing*, 2018.
    - Estado: aceptado
    - Índice de impacto (JCR 2017): 3.241
    - Área de conocimiento:
        - Computer Science, Artificial Intelligence. Ranking 27 / 132 - Q1

MV-kWNN: A novel multivariate and multi-output weighted nearest neighbours algorithm for big data time series forecasting

R. Talavera-Llames, R. Pérez-Chacón, A. Troncoso*, F. Martínez-Álvarez

Division of Computer Science, Universidad Pablo de Olavide, ES-41013 Seville, Spain

Abstract

This paper introduces a novel algorithm for big data time series forecasting. Its main novelty lies in its ability to deal with multivariate data, i.e. to consider multiple time series simultaneously, in order to make multi-output predictions. Real-world processes are typically characterised by several interrelated variables, and the future occurrence of certain time series cannot be explained without understanding the influence that other time series might have on the target time series. One key issue in the context of the multivariate analysis is to determine a priori whether exogenous variables must be included in the model or not. To deal with this, a correlation analysis is used to find a minimum correlation threshold that an exogenous time series must exhibit, in order to be beneficial. Furthermore, the proposed approach has been specifically designed to be used in the context of big data, thus making it possible to efficiently process very large time series. To evaluate the performance of the proposed approach we use data from Spanish electricity prices. Results have been compared to other multivariate approaches showing remarkable improvements both in terms of accuracy and execution time.

Keywords: big data, multivariate time series, forecasting

*Corresponding author

Email addresses: rltallla@upo.es (R. Talavera-Llames), rpercha@upo.es (R. Pérez-Chacón), atrolor@upo.es (A. Troncoso), fmaralv@upo.es (F. Martínez-Álvarez)

1. Introduction

Trying to predict the future has always fascinated humankind, and is considered a key topic in many fields. Not only are predictions used to forecast the weather [25]; but also in many other areas, ranging from everyday problems such as urban traffic jams [1] to avoiding natural disasters and their consequences [9] and even predicting breast cancer [21]. In fact, one of the fields that has attracted most attention in recent years is using predictive analysis to help decision making as this could save a considerable amount of money. For instance, works such as [36] helping governments to make better decisions and investments, and the importance of including weather forecasts is highlighted [16] to help farm managers to chose the right crop for increasing their profit.

However, to achieve high accuracy in a prediction, it is typically required to analyse not just one variable but several, therefore formulating the task into a multivariable prediction problem.

In this context, we propose a multi-purpose and multivariable weighted nearest neighbour (MV-kWNN) algorithm for forecasting time series. The algorithm is expected to work particularly well with time series with patterns in the historical data. Thus, chaotic and random time series may not be properly forecasted with the algorithm. This research stems from a previous work, where a univariable kWNN algorithm [38] was proposed, achieving a low mean relative error when it was applied to energy consumption in several buildings in a public university. Nevertheless, as is often the case, the forecast of a given target variable depends not only on the variable itself but also on some other variables. Consequently, better results will be achieved by taking these variables into consideration. Our study shows that, if these variables reach a certain degree of correlation, the prediction with the MV-kWNN algorithm will be improved.

Another relevant feature of the proposed algorithm lies in its ability to deal with big data time series. Over the last few years, since gathering information is no longer a problem, more and more data are being collected every day. In the context of time series, the Internet of Things (IoT) boom [14, 44] ([44] with more than 90 works) and the recent attraction to online user activities are some examples [27, 39]. In particular, for IoT, just one device monitoring a couple of parameters periodically could be affordable for classical implementations [42]. But when there are a considerable number of devices, the amount of information produced needs to be managed with a

new approach, as classical implementations are not implemented to deal with such a problem. Therefore, some researchers have begun to try to address big data time series [32].

New technologies have arisen to deal with high computational costs by distributing the data across computers in a cluster. One of the most well-known technologies is the Apache Hadoop open source software [12], based on the MapReduce paradigm [7], which distributes a dataset on the hard drives of the computers of a cluster, running the algorithm where the data are. However, when an iterative algorithm needs to be run, it could considerably increase the execution time. In this context, another framework has gone one step further. The Apache Spark open source software [45] distributes the data in the RAM memory of the computers of a cluster, making the execution considerable faster than when using Hadoop. The first steps of using the Apache Spark framework can be difficult to take, especially the cluster configuration. Therefore, some researchers have proposed methods to automate such a task [8].

For all reasons above, a multivariate-multi-output Spark based implementation to forecast time series with arbitrary horizon prediction has been carried out so that massive datasets can be processed. Data from Spanish electricity markets have been analysed. The reported results show that datasets of millions of samples can be efficiently processed. The linear relationship between the dataset size and the execution time confirms the suitability given its scalability of the proposed method for processing big data.

Synthetic datasets were generated to empirically analyse the a priori usefulness of the approach, by discovering a minimum correlation threshold between the exogenous variables and the target variable, that the exogenous variables must satisfy in order to be beneficial.

Furthermore, comparison to multivariate versions of random forests [15], artificial neural network (ANN) [13], and classical multivariate Box-Jenkins methods [20], such as autoregressive (ARX), autoregressive-moving-average (ARMAX) and autoregressive integrated moving average (ARIMAX) [3], confirm that the proposed algorithm is also better in terms of accuracy. Finally statistical tests have been applied to prove meaningful statistical differences among all the considered methods.

The rest of the paper is organised as follows. Section 2 reports relevant and related works. Section 3 introduces the methodology proposed in this paper. Section 4 describes how the method has been adapted to the big data context. The experimental setup carried out along with the study case show-

ing the most relevant results achieved and the scalability of the methodology are discussed in Section 5. Finally, the conclusions drawn from this research work are summarised in Section 6.

2. Related Work

The k nearest neighbour algorithm (kNN) is one of the most popular and important algorithms for data mining [43]. Its main purpose is to classify samples against a training set in a supervised manner, as the work [22] shows, where a distributed kNN algorithm is developed using the Apache Spark framework to classify different big datasets quickly and efficiently.

However, kNN-based approaches do not address high computational costs issues associated with their application to large datasets. This is due to the algorithms' need to calculate the distance between the new sample and every sample from the training dataset, and then to choose the k closest ones, which is computationally expensive for large datasets –namely $O(n \cdot log(n))$.

Despite its simplicity, the kNN algorithm provides a reliable solution to almost every kind of task. In [19] a kNN algorithm is implemented to carry out clustering on a large number amount of pictures, to try to find near duplicates. The algorithm is used as a previous step in an image processing problem. In particular, the authors manage the problem of finding approximate nearest neighbours for a repository of over one billion images (in the authors' words, the largest image set that has been processed in this way), and perform clustering based on these results. The only way presented to accomplish such a task was a parallel version of a state of art approximate nearest neighbour search algorithm, known as spill trees. Most of the work explained was concerned with efficiently finding the k nearest neighbours of points. To adapt this issue for clustering, the authors compute the k nearest neighbours for all images in the set and apply a threshold to drop images, which are considered too far apart. Although a scalability analysis is not made, the entire processing time is given, being less than 10 hours on the equivalent of 2000 CPUs.

In [37], the authors used a kNN algorithm for pattern recognition. In particular, the kNN is applied to a large image set to tag and classify them. The kNN has even been applied for malware detection as shown in [5]. The main objective of their research is to identify machines with malware infection communicating over HTTPS. To achieve this goal, they first perform a feature extraction from the proxy logs and then carry out a classification with a kNN

algorithm. For further information about fields in which the kNN has been successfully applied, please see the survey [26], where around 50 studies are analysed and classified.

The kNN algorithm has been sufficiently tested for time series prediction over small datasets related to energy. For example, in [4] a new technique for building energy modelling based on kernel regression using a kNN implementation is proposed. The results obtained are suitable although the kNN implementation is approximate to avoid high computation costs.

Additionally, the work in [4] compares several techniques, kNN among others, to try to predict appliance energy use and whole-home energy consumption, demonstrating that simple statistic algorithms achieve similar results than more complicated machine learning techniques with a lower computational cost.

In [6] a new algorithm, Maximum Length Weighted Nearest Neighbour (MLWNN), for predicting the electricity load in three countries is proposed. MLWNN is based on the WNN algorithm [41] which calculates the forecast depending on the distance to the nearest neighbours. The authors compared the predictions made with MLWNN and the predictions made with WNN, with a neural network [34] and with the PSF [24] algorithm which predicts based on similarity of pattern sequences. The research concludes that MLWNN accomplishes similar or better results than the other proposed methods.

Some other techniques have been developed in the last few years to forecast time series in the context of big data. In [31] the energy consumption in several buildings of a public university is predicted by applying a distributed k-means algorithm in Spark. The study in [40] shows that suitable accuracy predictions can be achieved by applying a deep learning algorithm to electricity consumption data in Spain. On the other hand, in [10] different scalable methods (decision tree, gradient boosted trees and random forest) are used to also predict the electricity consumption in Spain.

It is important to mention that often the prediction for a given variable depends not only on the previous values of the same variable, but also on other variables. By considering these additional exogenous variables, the accuracy may be improved. An example of multivariate prediction problem is considered in [17] - predicting the energy consumption of a building controlled by a building energy management system, taking into account temperature, dew point and solar radiation [17].

Another study, [46] aims to predict the company growth based on several

variables such as sales, assets or retained earnings among others. Due to the large number of input variables involved, the authors first had to select a smaller set of important input variables.

In [33], the accuracy of a univariate and multivariate methods for photovoltaic solar power forecasting is compared. The multivariate method uses weather data, in addition to the previous solar power. Several forecasting horizons are considered. The results showed that for very short-term horizons, the multivariate prediction is very similar to the univariate prediction. A possible explanation is that the weather changes are already reflected in the photovoltaic power data for very short-term horizons.

To the best of our knowledge, there are no further studies that have tried to address multivariate prediction for big data. This work intends to fill this gap and proposes a novel forecasting algorithm for big data multivariate time series.

3. Methodology

This section describes the proposed approach to forecast multivariate time series, in the context of big data. The approach is based on the general nearest neighbours procedure for time series forecasting [41] and, therefore, it follows a similar strategy, that can be summarised as follows:

1. Determine the number of nearest neighbours, k, which will be used.
2. Determine the length of the window composed of w samples preceding the target day.
3. Search for k nearest neighbours with length w in the historical data.
4. Calculate the estimation by averaging samples retrieved just after the k nearest neighbours. Additionally, since this is a weighted version, calculate the estimation by weighting the samples according to a given metric.

Given the multivariate nature of the proposal, these steps must be adapted. The remainder of this section provides a mathematical statement and formulation for the problem.

Let us suppose there are q variables indexed over time, that is, q time series forming a multivariate problem, formulated as follows:

$$
\begin{aligned}
V_1(t) &= \{v_1(t-L), \ldots, v_1(t-1), v_1(t)\} \\
V_2(t) &= \{v_2(t-L), \ldots, v_2(t-1), v_2(t)\} \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
V_q(t) &= \{v_q(t-L), \ldots, v_q(t-1), v_q(t)\}
\end{aligned}
$$

where L denotes the length of every time series. For simplicity, let us suppose all time series $V_1(t)$, $V_2(t)$,... , $V_q(t)$ have the same frequency of sampling. Nevertheless, the implementation of the MV-kWNN algorithm in Spark enables different frequencies of measurement for each time series.

Let h be the prediction horizon, i.e., the number of samples that must be predicted for every time series. Since q is the number of time series, the matrix P of dimensions $q \times h$ is composed of the total number of values to be predicted.

$$
\mathbf{P} = \begin{bmatrix}
v_1(t+1) & \ldots & v_1(t+h-1) & v_1(t+h) \\
v_2(t+1) & \ldots & v_2(t+h-1) & v_2(t+h) \\
\vdots & \ddots & \vdots & \vdots \\
v_q(t+1) & \ldots & v_q(t+h-1) & v_q(t+h)
\end{bmatrix}
$$

Let W be the l values preceding those aimed to be predicted for each time series such that:

$$
\mathbf{W} = \begin{bmatrix}
v_1(t-l) & \ldots & v_1(t-2) & v_1(t-1) \\
v_2(t-l) & \ldots & v_2(t-2) & v_2(t-1) \\
\vdots & \ddots & \vdots & \vdots \\
v_q(t-l) & \ldots & v_q(t-2) & v_q(t-1)
\end{bmatrix}
$$

The next step involves creating a distance matrix containing the distance between W and every possible matrix that can be formed throughout the historical data. Distance is calculated according to:

$$
D_{N_i} = \frac{1}{l} \sum_{j=1}^{l} d(W^j, N_i^j) \tag{1}
$$

where D_{N_i} represents the distance from W to the i-th closest neighbour N_i. This distance is calculated by a summation of the measure of the Euclidean

distance of every j-th column vector of W (W^j) to every j-th column vector of the neighbour N_i (N_i^j) and divided by the number of columns, l.

Subsequently, the k neighbours with the smallest distance are selected, and the weight of every neighbour is calculated with the formula defined as:

$$\alpha_i = \frac{1}{(D_{N_i})^2} \tag{2}$$

where α_i represents the weight of the i-th closest neighbour, inversely proportional to the distance.

Afterwards, in order to calculate the prediction, the h next values of every variable of every neighbour are selected forming the matrix R_{N_i}, which represents the next h real values of the i-th neighbour N_i.

Finally, the weight is applied to every R_{N_i} of the k closest neighbour and with the Equation (3), the prediction is made, resulting in a matrix $\widehat{P}$ of size q x h with the predictions for every time series.

$$\widehat{P} = \frac{1}{\sum_{i=1}^{k} \alpha_i} \sum_{i=1}^{k} \alpha_i R_{N_i} \tag{3}$$

4. Implementing the methodology in Spark

Working with Spark allows us to manage big data datasets and obtain results in a significantly shorter amount of time. In Spark, everything is performed using its own variables, called RDD (Resilient Distributed Datasets). These variables distribute the data in chunks over the RAM memory of every slave in a cluster. This is the key point of Spark, where all the computation is done using the RAM memory and not the hard drive; in other words, the execution of a programme is significantly faster in Spark than in other frameworks such as Hadoop that use the hard drive.

Figure 1 presents a diagram of the methodology proposed to apply the MV-kWNN algorithm. The first step of the algorithm is loading the datasets of the different time series into RDDs in Spark (Step 1). In the next step, the algorithm forms the matrix of w past values and its next h values for each time series, using the variables w_i and h_i (Step 2).

Subsequently, all matrices are joined into one matrix (Step 3). Then, the algorithm splits this matrix into training set (TRS) and test set (TES) (Step

4) selecting the w pattern composed of $w_1 + w_2 + w_3$ values, whose future needs to be predicted (Step 5).

From this point on, the MV-kWNN is properly applied to obtain the predictions. That is, it will compare the w pattern with the historical data in the matrix and will calculate the distances (Step 6). After that, the matrix is ordered in an ascending order and just the first k neighbours with the smallest distance to the pattern will be selected (Step 7). The algorithm will then take the h following values of the neighbours and will apply the weighted formula according to the distances. Next, the values are divided by the sum of the weights, resulting in the vectors with all predictions for all time series (Step 8). Then, the algorithm will stop and calculate the accuracy of the prediction for each time series.

It should be highlighted that all these operations are done in Spark over the distributed RDD variables.

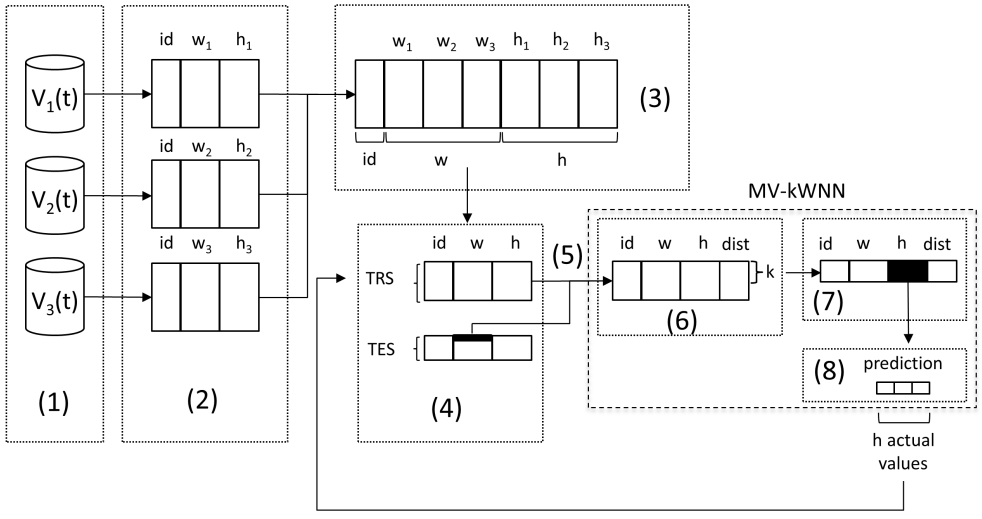


Figure 1: A diagram of the proposed methodology.

In more detail, let us suppose that a multivariate time series composed of three time series, $V_1(t)$, $V_2(t)$ and $V_3(t)$, needs a prediction for each one. Let us also suppose, for simplicity, that the values w and h are expressed in hours. As the time series composing of the multivariate time series could have different frequency of sampling, the algorithm creates the windows w_i and prediction horizon h_i for each time series such that the same number of

past hours are considered and the same number of hours are predicted for each time series.

Figure 2 (a) shows how w_i and h_i values are computed and how a matrix of w_i past values and its next h_i values for each time series is formed, supposing that w represents two hours and h represents one hour. For the first time series $V_1(t)$ there are two samples per hour ($v1(0)$ and $v_1(1)$), for the second time series $V_2(t)$, four samples correspond to one hour ($v_2(0)$, $v_2(1)$, $v_2(2)$, $v_2(3)$) and just one sample corresponds to one hour ($v_3(0)$) for the third time series $V_3(t)$.

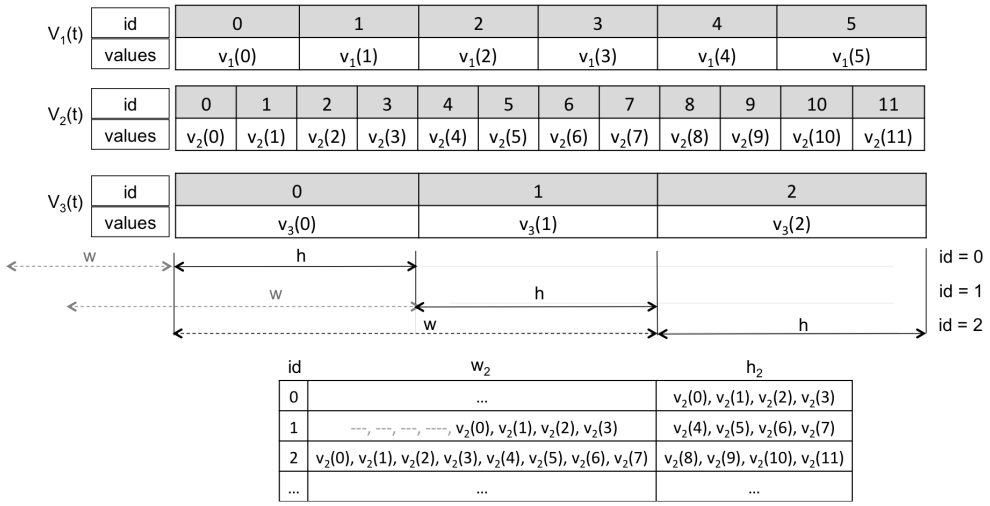
In general, each w_i will be the number of samples per hour multiplied by the window w, while each h_i will be the number of samples per hour multiplied by the prediction horizon h. In the figure:

- For $V_1(t)$: $w_1 = 2 * w = 4$ and $h_1 = 2 * h = 2$.
- For $V_2(t)$: $w_2 = 4 * w = 8$ and $h_2 = 4 * h = 4$.
- For $V_3(t)$: $w_3 = 1 * w = 2$ and $h_3 = 1 * h = 1$.

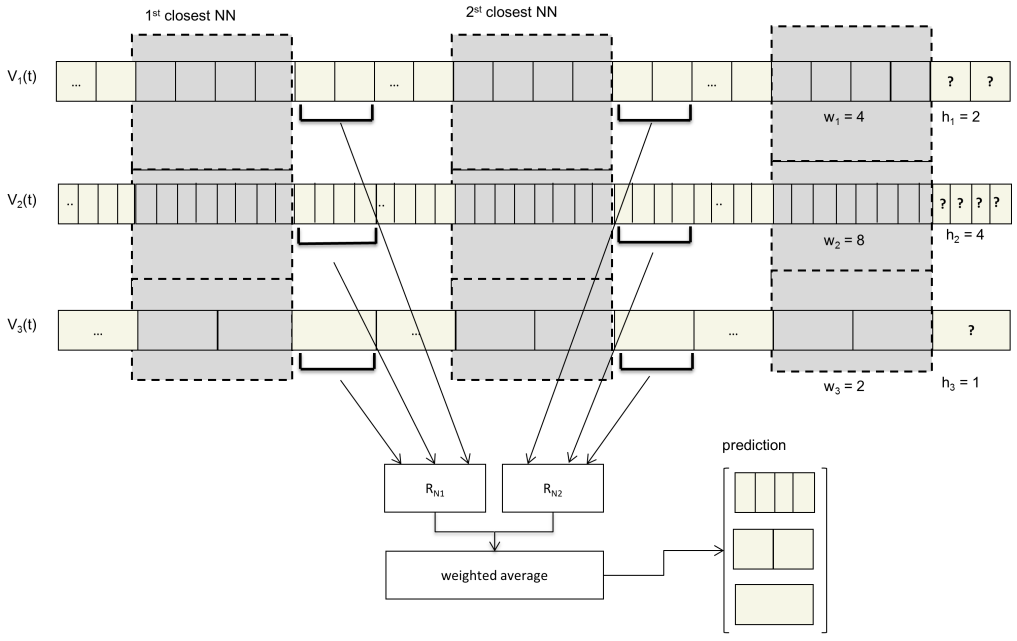
The MV-kWNN algorithm works as it is shown in Figure 2 (b). In this figure, h_1, h_2 and h_3 values have to be predicted and w_1, w_2 and w_3 values represent the windows of past values for each time series, respectively. First, the windows w_1, w_2 and w_3 to search for its k nearest neighbours needs to be selected for each time series ($k = 2$ in this figure). Afterwards, the algorithm will search for the k nearest neighbours to the array formed for $w_1 + w_2 + w_3$ values. The next h values of every time series that formed the closest neighbours will be selected, creating the matrices R_{N_1} and R_{N_2}. Finally, by applying a weighted average combination according to the distance between closet neighbours and the window of $w_1 + w_2 + w_3$ values, the prediction for each variable will be made.

For a better understanding of how the algorithm works, an appendix (Section 7) with a numerical example has been included. Figure 14(a) shows a test example with three time series, V_1, V_2 and V_3, all of them with values in the range [0,1] after been normalised. The number of values that needs to be predicted for each time series is two (h=2). The size of a window of past values is set to four (w=4). In this case, the last four values of each time series will be selected to form the $wPattern$ to search for its closest neighbours.

After that, the values of the three time series are grouped in windows of h values and its w previous values, as columns $w's$ and $h's$ show in Figure 14(b).



(a) Forming matrix of w past values and its next h values.



(b) Selecting k closest neighbours and calculating the prediction.

Figure 2: A test example of the MV-kWNN algorithm with time series with different frequency of sampling.

All of these w windows are considered to be neighbours of the $wPattern$. To choose just the k closest neighbours, two in this case, the distance to each w window to the $wPattern$ is calculated. Once the neighbours have been selected, their h windows are considered to calculate the prediction. In order to do so, first, each value of the h window is multiplied by the weight, that is $1/dist^2$. Secondly, to form a unique window of h size, the values in the same position of the $h's$ windows are added. The weights are also aggregated. All these steps can be seen in Figure 14(c).

Finally, to make the prediction, each value of the previous windows is divided by the sum of the weights, resulting in the final forecast. In this case, the first two values correspond to the prediction of the V_1 time series, the next two values are the prediction of the V_2 time series and the final two values are the prediction of the V_3 time series.

5. Experimentation and results

This section reports the experimentation carried out and the results achieved by the application of MV-kWNN to multivariate time series. First, the experimental setup is commented in Section 5.1. Particularly, the metrics used to assess the performance of the method are formulated along with a numerical analysis focused on finding correlation thresholds among the multiple time series and description and analysis of the dataset used for the experimentation. Section 5.2 described a successful case study of a real-world problem, comparing the results with other multivariate methods and including statistical tests. Finally, a scalability analysis is conducted in Section 5.3 to show that the method is suitable to be used in the context of big data time series.

5.1. Experimental setup

5.1.1. Quality parameters

To evaluate the accuracy of the predictions made by the MV-kWNN algorithm, the mean relative error (MRE), the mean error relative to the mean of the test set (MMRE) and the mean absolute error (MAE) along with bias (BIAS) and root squared error (RMSE) will be used. The formulas are shown below:

$$MRE = 100 \cdot \frac{1}{n} \sum_{i=1}^{n} \frac{|a_i - p_i|}{a_i} \tag{4}$$

$$MMRE = 100 \cdot \frac{1}{n}\sum_{i=1}^{n}\frac{|a_i - p_i|}{\overline{x_{TES}}} \tag{5}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|a_i - p_i| \tag{6}$$

$$BIAS = \frac{1}{n}\sum_{i=1}^{n}(a_i - p_i) \tag{7}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - p_i)^2} \tag{8}$$

where n represents the total number of predictions made, a_i represents one actual value, p_i represents the prediction for that actual value and $\overline{x_{TES}}$ represents the mean of the test set. It should also be mentioned that if a time series contains zero values $MMRE$ will be used instead of MRE (although MRE will be always mentioned for simplicity). This is because it will be not possible to divide the difference between the prediction and the actual value by zero. Instead, it will be divided by the test set mean.

The MAE express the exact amount of electricity and the exact amount of money above or below predictions. This information is of a significant importance for electricity companies in order to plan their energy production. As a study of the literature in electricity time series forecasting shows, MAE and MRE are the most accepted parameters to measure the accuracy of the predictions [41, 18, 2]. Nevertheless, two more standard measures have been included, BIAS and RMSE. Note that to express the MRE in percentage, it has been multiplied by 100.

5.1.2. Numerical study of the usefulness of MV-kWNN on simulated data

Before MV-kWNN is applied, how much information the exogenous time series may add to the multivariate problem must be determined. This is necessary as using multiple variables to formulate a multivariate problem does not necessarily lead to improvements in accuracy. To determine if an exogenous variable will be beneficial, we propose an analysis based on the correlation.

MV-kWNN is based on the discovery of nearest neighbours over the historical data, therefore, if exogenous time series do not exhibit any correlation with future values of the target time series, the error should increase.

Table 1: Synthetic datasets generated with different correlations with the prediction horizon.

Target time series	Exogenous time series				
$Y_1^0(t)$	$X_{1,1}^0(t)$	$X_{1,2}^{0.25}(t)$	$X_{1,3}^{0.50}(t)$	$X_{1,4}^{0.75}(t)$	$X_{1,5}^1(t)$
$Y_2^{0.25}(t)$	$X_{2,1}^0(t)$	$X_{2,2}^{0.25}(t)$	$X_{2,3}^{0.50}(t)$	$X_{2,4}^{0.75}(t)$	$X_{2,5}^1(t)$
$Y_3^{0.50}(t)$	$X_{3,1}^0(t)$	$X_{3,2}^{0.25}(t)$	$X_{3,3}^{0.50}(t)$	$X_{3,4}^{0.75}(t)$	$X_{3,5}^1(t)$
$Y_4^{0.75}(t)$	$X_{4,1}^0(t)$	$X_{4,2}^{0.25}(t)$	$X_{4,3}^{0.50}(t)$	$X_{4,4}^{0.75}(t)$	$X_{4,5}^1(t)$
$Y_5^1(t)$	$X_{5,1}^0(t)$	$X_{5,2}^{0.25}(t)$	$X_{5,3}^{0.50}(t)$	$X_{5,4}^{0.75}(t)$	$X_{5,5}^1(t)$

To determine when MV-kWNN is suitable for solving multivariate problems, synthetic datasets have been generated. A total of five time series, $Y_1(t)$, $Y_2(t)$, $Y_3(t)$, $Y_4(t)$ and $Y_5(t)$, have been generated with correlations $c = \{0, 0.25, 0.5, 0.75, 1\}$ with its future value, respectively. That is, they have been generated so that sample t has correlation c with sample $t + h$, where $h = 4$ is the prediction horizon in this analysis. The generated time series are normalised and all values range from 0 to 1. These correlated series have been generated by means of MATLAB's *randwithcorr* function, which generates a random time series with a specific correlation to a given time series.

The next step includes combining each of these time series with another time series so that sample t has correlation c with sample $t + h$ from the previously generated time series. In other words, 25 exogenous time series $X_{i,j}(t)$ have been generated, as shown in Table 1, where $Y_i^\rho(t)$ denotes a time series $Y_i(t)$ with correlation ρ with $Y_i(t + h)$ and $X_{i,j}^\rho(t)$ is a time series with correlation ρ between $X_{i,j}(t)$ and $Y_i(t + h)$. Thus, for instance, $Y_2^{0.25}(t)$ is the time series generated by using *randwithcorr* function from the time series $Y_2(t + h)$ and coefficient of correlation 0.25.

It needs to be mentioned that for every experiment, 70% of the total number of samples has been considered to train the model and the 30% remaining to test the model. Tables 2 and 3 show the associated MRE and MAE, respectively, for the experimental setup described above. Since the conclusions are similar, we only discuss the values for Table 2.

As expected, it can be seen that the errors are constantly decreasing as long as the correlation between $X_{i,j}(t)$ exogenous variables and $Y_i(t + h)$ increases. It is worth noting that if the exogenous variable shows a greater correlation with $Y_i(t + h)$ than the target variable itself, the error decreases acutely (see the errors above the diagonal). However, if the opposite situation

Table 2: MRE for synthetic datasets with different correlations.

Target	Target vs. Exogenous				
$Y_1^0(t)$	77.55	59.23	38.01	23.09	14.43
$Y_2^{0.25}(t)$	59.23	55.51	34.02	22.17	14.77
$Y_3^{0.50}(t)$	38.01	34.02	33.93	20.46	12.86
$Y_4^{0.75}(t)$	23.09	22.17	20.46	19.62	11.44
$Y_5^1(t)$	14.43	14.77	12.86	11.44	6.70

Table 3: MAE for synthetic datasets with different correlations.

Target	Target vs. Exogenous				
$Y_1^0(t)$	10.58	8.86	6.69	4.99	3.29
$Y_2^{0.25}(t)$	8.86	8.52	6.33	4.80	3.25
$Y_3^{0.50}(t)$	6.69	6.33	6.23	4.30	2.87
$Y_4^{0.75}(t)$	4.99	4.80	4.30	4.32	2.50
$Y_5^1(t)$	3.29	3.25	2.87	2.50	1.49

is encountered (the correlation with $Y_i(t+h)$ is less than that of the target variable) then the error increases but very slightly (see errors below the diagonal). That is, MV-kWNN seems quite robust in terms of accuracy even if the exogenous variables are not properly selected.

5.1.3. Datasets description

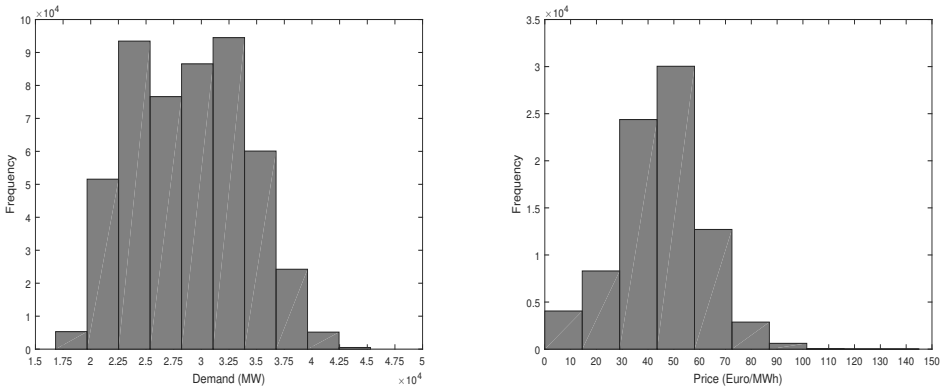
This section is devoted to analysis and explain the datasets used in the experimentation phase. Hence, data from the electricity Spanish market has been retrieved and studied. In particular, electricity prices and electricity demand since it is well-known that prices and demand are related to one another [23]. Data has been collected from the OMIE website (`www.omie.es`) from 2007 to 2016, with hourly values for the prices (expressed in Euro/MWh) and 10-minute values for the electricity demand (expressed in MW).

Table 4 shows a statistical analysis of the time series, with the number of samples, mean, standard deviation, minimum and maximum.

Figure 3 shows the distribution of the datasets. As it can be seen, for the electricity demand, almost all the values are concentrated between 20,000 and 37,500 MW with little amount of small and high values. On the other hand, for the electricity prices, almost all the values are concentrated between 25 and 65 Euro/MWh, with a significant number of small values and few very high ones.

Table 4: Statistical analysis of the datasets

	Demand	Price
Number of samples	497,952	82,992
Mean	28,897.40	44,94
Standard Deviation	5,058.89	16.72
Minimum	16,824	0
Maximum	45,295	145



(a) Distribution of the electricity demand. (b) Distribution of the electricity prices.

Figure 3: Distribution of the datasets.

5.2. Study case: electricity prices and demand in the Spanish market

This section reports results for a real-world problem (electricity prices and demand). To make the predictions, the model has been trained with the 70% of the total number of samples, using the remaining 30% for testing the accuracy of such predictions.

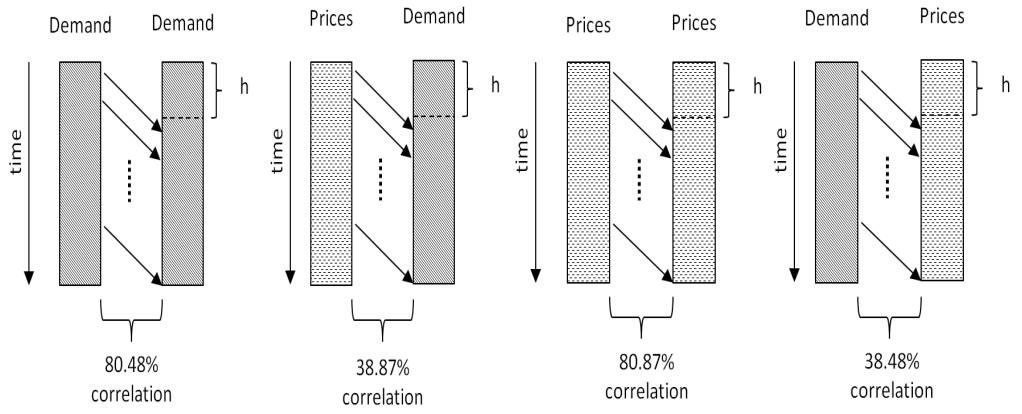
A prediction horizon of 4 hours and a window of past values of 24 hours have been set for the MV-kNN algorithm. For these experiments, the number of k nearest neighbours has been set to 4. A study of the influence of the size of the k parameter is analysed in subsection 5.2.1

It needs to be pointed out that values of both input datasets have been normalised dividing by the maximum value of each dataset, finally ranging from 0 to 1. This is due to values for the electricity demand originally range from 16,824 to 45,295 while values for the electricity prices range from 0 to 145. Hence, normalising both datasets makes that each value is considered

to be equally important.

We firstly discuss the correlations values between the time series:

- On one hand, if the electricity demand is the target variable, then its correlation with the demand shifted 24 values, that is with its own immediate future, is 80.48%. However, the correlation between the electricity prices and the immediate future of the electricity demand is 38.87%. Both correlations can be seen in Figure 4 (a).
- On the other hand, if the electricity price is the target variable, its correlation with the prices shifted 24, that is with its own immediate future, values is 80.87%. However, the correlation between the electricity demand and the immediate future of the prices is 38.48%. Both correlations can be seen in Figure 4 (b).



(a) Correlations when electricity demand is the target variable.

(b) Correlations when electricity prices is the target variable.

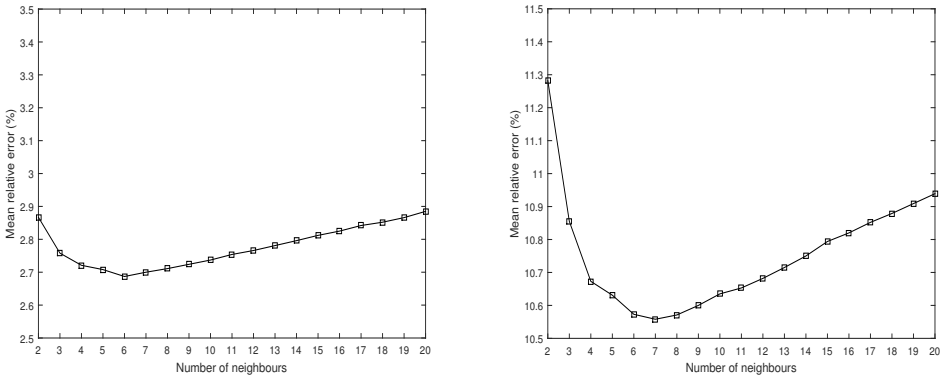
Figure 4: Correlations between the time series.

5.2.1. Anaylis of the number of neighbours

In this subsection, the number of neighbours, k, to be selected for the algorithm is discussed. k ranging from 2 to 20 have been considered.

Figure 5 shows the MRE obtained for the predictions of the electricity demand and for the electricity prices with the different number of neighbours. As it can be seen, the error keeps constant along with the number of neighbours for both predictions. In particular, MRE for the electricity demand

ranges from 2.69% to 2.86%, while MRE for the electricity prices range from 10.56% to 11.28%. Less than 1% difference between the maximum and the minimum values.



(a) MRE for the electricity demand.

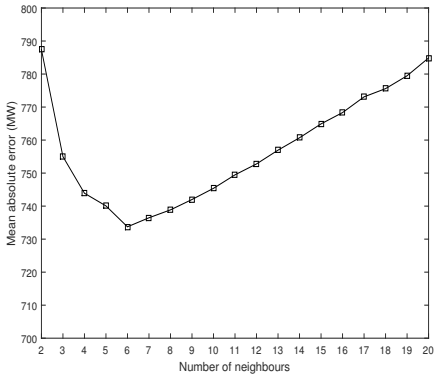
(b) MRE for the electricity prices.

Figure 5: MRE for the electricity demand and the electricity prices with different neighbours

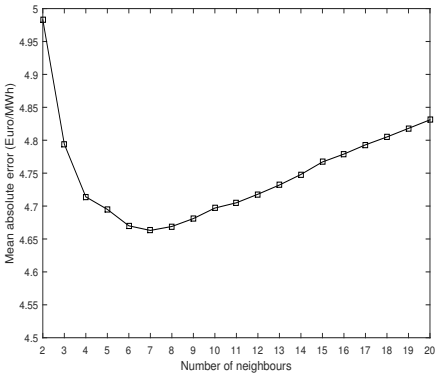
Figure 6 shows the MAE obtained for the predictions of the electricity demand and for the electricity prices with the different number of neighbours. The former expressed in MW and the latter expressed in Euro/MWh. As previously mentioned for the MRE, the error keeps constant along with the number of neighbours for both predictions. In particular, MAE for the electricity demand ranges from 733.73 to 787.42, while MAE for the electricity prices range from 4.66 to 4.98. In summary, 53.68 MW of difference between the maximum and the minimum values in case of the electricity demand and 0.3 Euro/MWh of difference between the maximum and the minimum values in case of the electricity prices.

5.2.2. Analysis of the electricity demand prediction

In this subsection, the accuracy of the algorithm is computed for the electricity demand. The MRE of the prediction for the test set has been 2.72%. In the following graphs, the results will be analysed. Figure 7 presents the mean relative error of the prediction for each month. Figure 7 (a) shows the MRE of every prediction per month. From that figure, it could be thought that there are many outliers errors. However, these errors do not represent a significant number due to the fact than 75% of all errors in every month are



(a) MAE for the electricity demand.



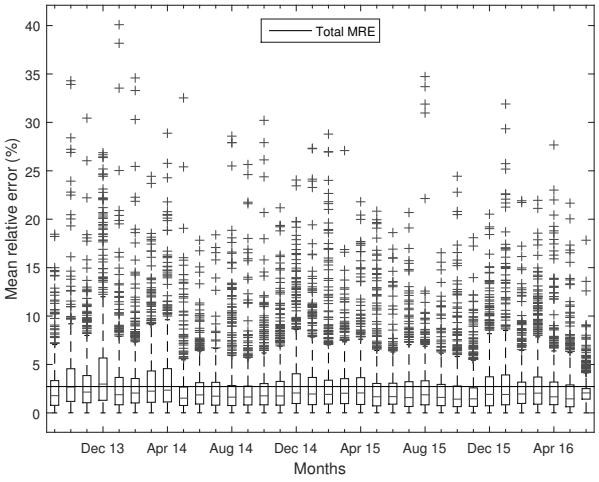
(b) MAE for the electricity prices.

Figure 6: MAE for the electricity demand and the electricity prices with different neighbours

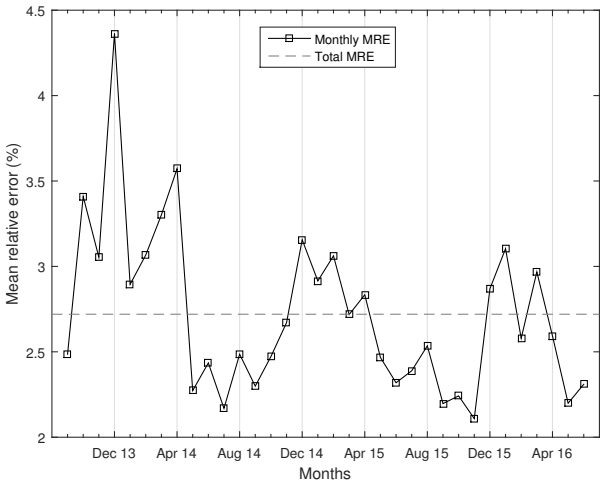
below 5.5%. From Figure 7 (b) it can be seen that the highest error is 4.36% in December 2013, while the lowest MRE is 2.11% in November 2015. Both months will be analysed below.

Figure 8 represents the months with the lowest and highest MRE. Particularly, Figure 8 (a) represents November 2015, the best case with the MRE for each day remaining steady around the MRE of the whole month. Day 19 of the month reaches the best accuracy with MRE close to zero (0.55%) meanwhile day 22 achieves the worst MRE, with 5.19%. On the other hand, Figure 8 (b) represents December 2013, the worst case with the MRE for each day varying significantly over the month. Predictions achieves greater erros in days 30, 7 and 9, with 10.34%, 8.83% and 8.00% respectively. Lower errors are reached in days 18 and 19 with 1.20% and 1.6% respectively.

Figure 9 shows the real energy demand against those predicted for the best and worst days in terms of mean relative error and also the day with similar MRE to the total prediction. Figure 9(a) represents the day with the lowest MRE (0.40%), 23^{rd} July 2015. As can be seen both the real and the predicted values are almost alike. On the other hand, Figure 9(b) represents the day with the highest MRE (11.68%), 10^{th} February 2014. The algorithm at the beginning of the day predicts very accurately, until 6.00 AM, where it can be noted that the prediction made follows the real energy demand but shifted some values. Finally, Figure 9(c) shows the day with similar MRE as the total prediction (2.72%), 28^{th} February 2015. Again the algorithm at the



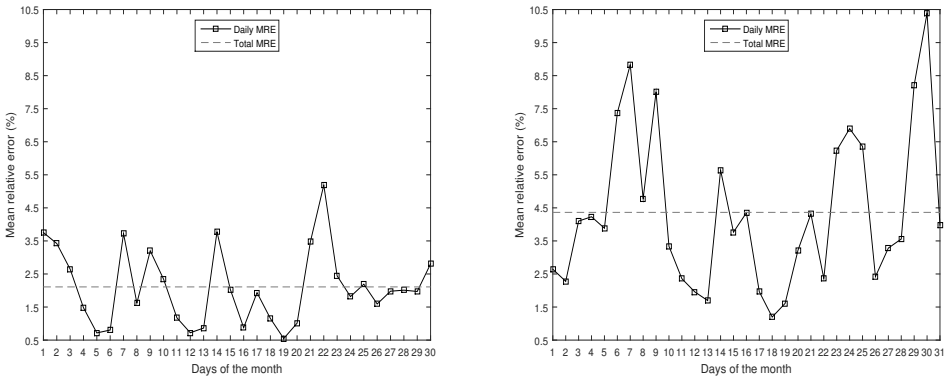
(a) MRE along with the standard deviation for the test set for each month.



(b) MRE of the prediction for the test set for each month.

Figure 7: Monthly relative prediction errors for the electricity demand.

beginning of the day predicts very accurately, until 7.00 AM, where it raises the prediction faster than the real values. Then by 9.00 AM the algorithm predicts adjusted to the real values and following the curve of energy demand



(a) Month with the lowest MRE. (b) Month with the highest MRE.

Figure 8: MRE for the best and worst month for the electricity demand.

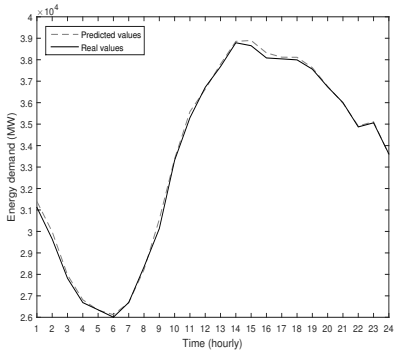
through the day.

5.2.3. Analysis of the electricity prices prediction

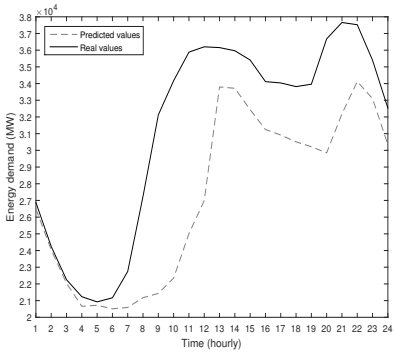
In this subsection, the accuracy of the algorithm is computed for the electricity prices. The MRE of the prediction for the test set has been 10.67%. In the following graphs, the results will be analysed. Figure 10 presents the mean relative error of the prediction for each month. Figure 10 (a) shows the MRE of every prediction per month. The 75% of all errors in every month are below 20%, except months December 2013, January 2014 and February 2014, months with the highest errors in the predictions as can also be seen in Figure 10 (b). December 2013 will be analysed later. Figure 10 (b) also shows that the lowest MRE is 6.46% in June 2015. This month will be analysed below.

Figure 11 represents the months with the lowest and highest MRE. June 2015 represents the best case with the MRE for each day remaining steady around the MRE of the whole month. In fact, day 29 of the month achieves the worst MRE, with 11.19% meanwhile day 9 achieves the best MRE with 2.78%. On the other hand, December 2013 represents the worst case with the MRE for each day varying significantly over the month. It achieves greater errors in days 20 and 24 with 45.29%, and 44.29% respectively. Day 20 will be analysed in detail below. Lower errors are achieved in days 25 and 6 with 6.76% and 9.88% respectively.

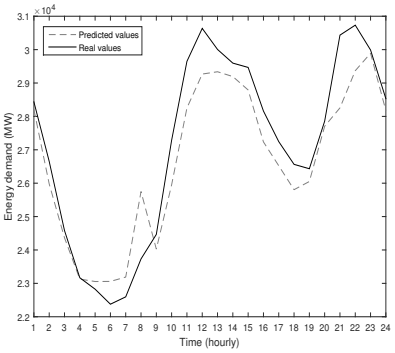
Figure 12 shows the real energy prices against those predicted for the best



(a) Day with the lowest MRE.



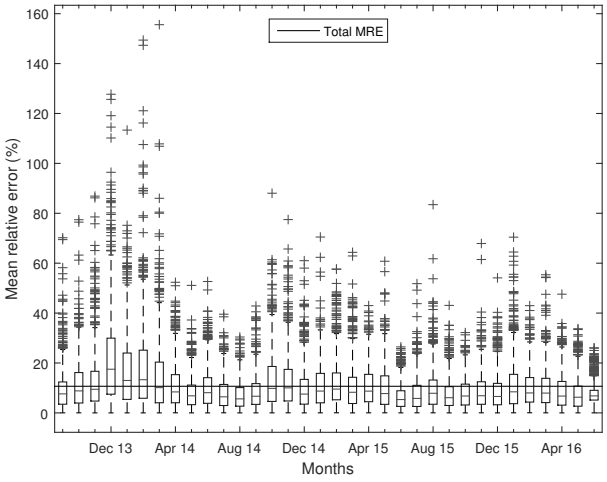
(b) Day with the highest MRE.



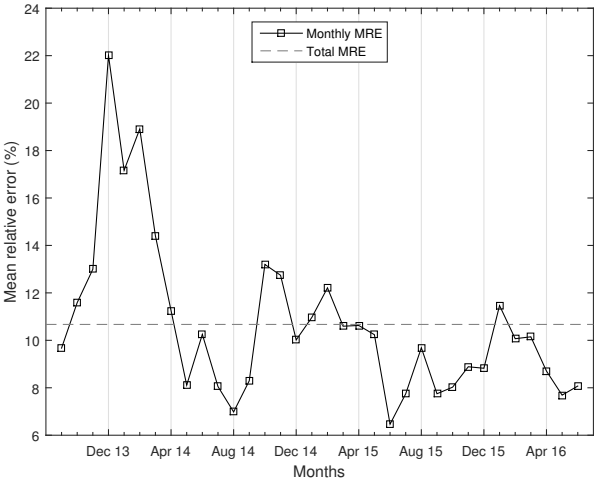
(c) Day with similar MRE as the total prediction.

Figure 9: Electricity demand. MRE for the days with the best, the worst and similar MRE as the total prediction.

and worst days in terms of mean relative error and also the day with similar MRE to the total prediction. Figure 12(a) represents the day with the lowest MRE (2.02%), 16^{th} April 2014. As can be seen both the real and the predicted values are very similar, with just some differences around 13.00 time, when the prices decrease a little bit faster than what the algorithm predicts. On the other hand, Figure 12(b) represents the day with the highest MRE (45.29%), 20^{th} December 2013. In this case, the algorithm predicts the same behaviour of the curve of prices, reaching the lowest prices at the same time, 5.00 AM. But the real values decrease until 0, what gives the greater differences in this prediction. It also did not predict the peak in the prices at 11.00 AM, but



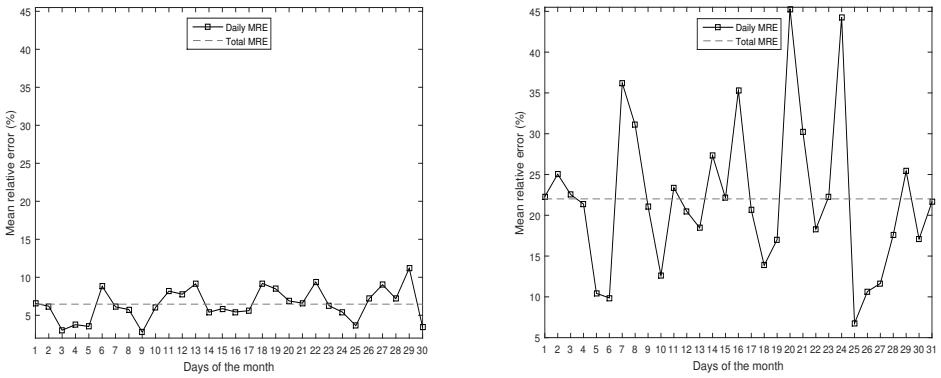
(a) MRE along with the standard deviation for the test set for each month.



(b) MRE of the prediction for the test set for each month.

Figure 10: Monthly relative prediction errors for the electricity prices.

even by the end of the day simulates the same curve of prices but shifted with lower values. Finally, Figure 12(c) shows the day with similar MRE as the total prediction (10.67%), 22^{nd} May 2014. The algorithm, in this case,



(a) Month with the lowest MRE. (b) Month with the highest MRE.

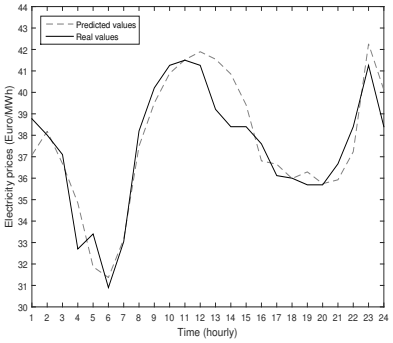
Figure 11: MRE for the best and worst month for the electricity prices.

predicts the same curve of prices, but again shift some values over the real ones until 7.00 AM, and shift some values below the real ones from 9.00 AM until the end of the day.

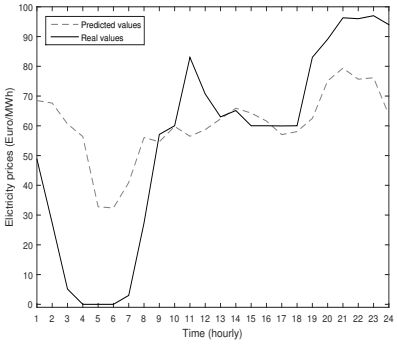
5.2.4. Comparison with others multivariate algorithms

The results of MV-kWNN have been compared to standard and nonlinear multivariate forecasting algorithms. Namely, artificial neural networks (MV-ANN) [13] and random forests (MV-RF) [15] using R packages described in [28, 29]. As for the classical methods, results are compared to multivariate versions of AR, ARMA and ARIMA [20]. In particular, ARX, ARMAX and ARIMAX MATLAB's implementations have been used, which are developments of Box-Jenkins models [3]. All multivariate versions have been configured so that comparisons are fair (same training/test sets, same prediction horizons). However, since these models do not allow different sampling values for each time series, the electricity demand time series has been resampled so that both demand and prices are on the same basis (hourly). We used a prediction horizon of 4 hours and a window of past values set to 24 hours. As for the particular configuration of the classical methods, they have been automatically selected by the MATLAB's functions we have used. In particular ARX (1), ARMAX (1,1) and ARIMAX (0,1,1) have been chosen.

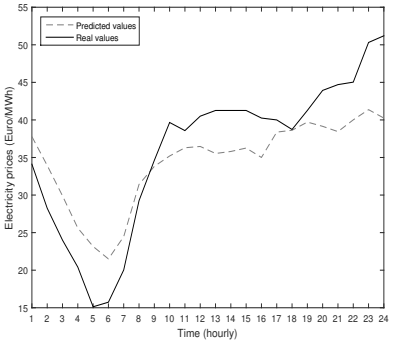
It also needs to be mentioned the settings for both MV-ANN and MV-RF, as they allow different configurations:



(a) Day with the lowest MRE.



(b) Day with the highest MRE.



(c) Day with similar MRE as the total prediction.

Figure 12: Electricity prices. MRE for the days with the best, the worst and similar MRE as the total prediction.

- MV-ANN:
  - To train the neural network, the backpropagation algorithm (BPNN) was used. BPNN is a very popular and very successful algorithm for electricity load forecasting [13].
  - The number of neurons tested in the hidden layer varies from 1 to 28. Since a specific rule to choose the number of neurons in the hidden layer is the mean of input and output neurons, in our case 14 neurons because h=4 and w=24, this upper bound (2x14) is particularly suitable for this problem.

  - The weights were initialised randomly, which is very common strategy [35].
  - The differentiable function that was used for the calculation of the error was the sum of squared errors, which is very common strategy [35].
  - The differentiable function considered for smoothing the result of the cross product of the covariate or neurons and the weights were two: the logistic function and the tangent hyperbolicus.These are the only two possible functions that the R package offers.
  - The maximum steps for the training of the neural network considered were: 1E+05, 1E+06 and 1E+07. Values above or below those mentioned before made the computation of the dataset impossible for the MV-ANN, due to the amount of data.
  - The threshold for the partial derivatives of the error function as stopping criteria considered were: 0.5, 0.6 and 0.7. Values above or below those mentioned before made the computation of the dataset impossible for the MV-ANN, due to the amount of data.

  The best results were obtained with the logistic function as the differentiable function, 28 as the number of neurons, 1E+06 as the maximum steps and a threshold of 0.5.

- MV-RF: The number of trees and the maximum depth are the main inputs for random forests.
  - The depth level tested were 4 and 8.
  - The number of trees tested were 50, 75 and 100.

  Values above those mentioned before made the computation of the dataset impossible for the MV-RF, due to the amount of data. The best results were obtained with a depth level of 4 and 100 trees.

Tables 5 and 6 summarises MRE, MAE, BIAS and RMSE for MV-kWNN, MV-ANN, MV-RF, ARX, ARMAX and ARIMAX multivariate models for the electricity demand and electricity prices. In the case of the prices time series, the MRE is computed with respect to the mean of the test set instead of the actual value since the prices can be close to zero [41]. It can be seen that MV-kWNN outperforms all models reaching very competitive errors in terms

of MRE, MAE and RMSE. As it can be noted in Table 5 for the electricity demand all methods obtain underestimated forecasts but the ARIMAX that has a positive bias. On the other hand, in Table 6 for the electricity prices, it can be seen the MV-kWNN, MV-RF and ARMAX have a bias very close to zero.

Table 5: Comparative results for electricity demand time series.

	Demand			
Model	MRE (%)	MAE (MW)	BIAS	RMSE
MV-kWNN	2.72	743.90	-99.00	1105.91
MV-ANN	2.87	779.72	-98.20	1018.30
MV-RF	3.44	927.27	-160.16	1235.04
ARX	8.06	2179.67	-58.80	2971.15
ARMAX	6.68	1798.65	-59.74	2499.58
ARIMAX	5.51	1529.75	176.39	2210.00

Table 6: Comparative results for electricity prices time series.

	Prices			
Model	MRE (%)	MAE (Euro/MWh)	BIAS	RMSE
MV-kWNN	10.67	4.71	-0.05	6.62
MV-ANN	10.96	4.79	-0.75	4.35
MV-RF	10.78	4.76	0.05	6.69
ARX	13.71	6.05	0.10	8.78
ARMAX	12.67	5.60	-0.04	8.15
ARIMAX	12.13	5.36	0.14	8.23

5.2.5. Statistical test with other multivariate algorithms

This section is devoted to evaluate the significance of the new approach, following the non-parametric procedures discussed in [11], carrying out a statistical test. The steps follows are explained below:

1. Five synthetic datasets, whose values range from 0 to 1, were generated. The total number of instances in each dataset was the same as in the electricity prices.
2. The synthetic datasets were combined in groups of two without repetitions resulting in ten experiments.

Table 7: Friedman Test ranking of the models.

Model	Position in the ranking
MV-kWNN	1.5909
ARX	2.9545
ARIMAX	3.5909
ARMAX	3.8182
MV-ANN	4.2273
MV-RF	4.8182

3. The MRE was considered as the quality meter to be measured and collected in every experiment.
4. All the multivariate algorithms were considered for the experiments (MV-kWNN, MV-ANN, MV-RF, ARX, ARMAX, ARIMAX)
5. Previous results from forecasting electricity demand and electricity prices were also considered, the MRE in this case. What in summary made a total of 11 experiments.

The statistical test has been done with the STATservice software [30]. This software allows for performing statistical analysis directly online. Firstly, it was tested if the distribution of the variables were normal. As it was not, a non parametric test was selected. Secondly, as the number of variables to be tested were greater than two, a multiple comparison test was chosen. Finally, as the number of the variables to be tested were not lower than six, a Friedman Test with Control estimation was conducted. The ranking after applying Friedman Test is shown in Table 7. As it can be seen, the MV-kWNN was ranked as the first one. The p-value is $0.0000 < 0.05$, therefore the null hypothesis is rejected.

Subsequently, a pairwise posthoc test is carried out as it can be seen in Table 8, comparing the algorithms with the best performing one, MV-kWNN. As Holm's procedure rejects those hypotheses that have a p-value$<=$0.05, the null hypothesis is rejected for all cases. Thus, the differences between MV-kWNN and the remaining algorithms are significant.

5.3. Scalability

This section is included in order to highlight the scalability of the proposed approach. It shows that the execution time increases linearly with increasing the data size. This fact is especially desirable for big data time

Table 8: Pairwise PostHoc Test

Model	p-value	z	Holm
MV-RF	0.0000	5.7213	0.0100
MV-ANN	0.0000	4.6738	0.0125
ARMAX	0.0001	3.9485	0.0167
ARIMAX	0.0004	3.5456	0.0250
ARX	0.0156	2.4175	0.0500

series, since it proves the applicability of our algorithm to this type of data. But first it needs to be mentioned the complexity of the algorithm. Finding the closest neighbour of just one instance of the TES from a TRS would be $O(n \cdot D)$, being n the number of instances of the training and D the number of features. Therefore, finding the k nearest neighbours would have a greater computational cost as the distances calculated needs to be sorted. This adds another complexity resulting in $O(n \cdot log(n))$. The process starts all over again for each instance of the test. Hence, the high computational cost of the kWNN makes impossible to use it when dealing with big data, justifying thus the need of developing a distributed kNN algorithm in order to obtain a scalable forecasting algorithm.

The experiments have been carried out on a cluster composed of three nodes, one master and two slaves, physically located at Data Science & Big Data Laboratory, Pablo de Olavide University. The nodes in every cluster are made up of the following features:

Table 9: Features of the nodes in the cluster.

Hardware	
- Processors	Intel(R) Core(TM) i7-5820K CPU
- Cores	6 (12 threads)
- Clock speed	2.13 GHz
- Cache	16 MB
- Network	Gigabit (1 Gb/s)
- RAM	16 GB
Software	
- Framework	Apache Spark 1.6.1
- OS	Ubuntu 16.04.1 LTS

Figure 13 illustrates execution times to calculate a prediction. In partic-

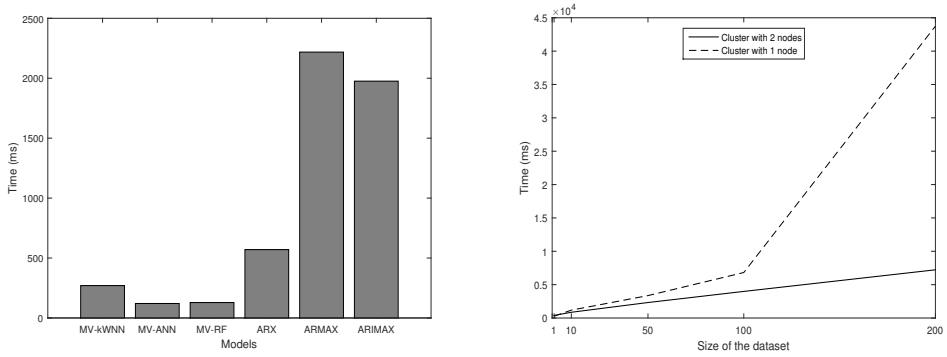
ular, Figure 13(a) shows execution times to calculate a prediction of every model tested in this paper. As the other models could not manage the bigger size of the datasets and are not prepared for being executed in a cluster, all the results shown in the figure were collected under the same conditions. That is, the size of the dataset was set to 1 (the original size of the dataset) and they were executed on just one machine of the cluster. It can be seen that MV-ANN and MV-RF were the faster ones, with a slight difference from the MV-kWNN. The slowest one was the ARMAX.

On the other hand, Figure 13(b) shows execution times to calculated a prediction compared with data size and different number of machines in the cluster. To obtain these results, the original prices time series has been replicated by up to 200 times with up to 16,593,600 samples. As mentioned before, just the MV-kWNN could be analysed for this test as only this algorithm could manage such amount of data. Firstly, it can be noticed that the execution time with one and two machines in the cluster are very alike for 1, 10 and 50 times the size of the original dataset. However, when the size of the dataset begins to be significant, with 100 times and 200 times the size of the dataset, the execution times are notably different. Secondly, focus on the execution times with two machines in a cluster, it can be seen that when the size of the dataset grows, the time to calculate a prediction increases in a linearly. In summary, the MV-kWNN algorithm is ready to manage big data datasets and the execution times will decrease with the number of computer in a cluster.

6. Conclusions

In this paper, we proposed a multi-purpose and multivariate MV-kWNN algorithm for big data time series forecasting which is expected to perform particularly well with time series with patterns in the historical data. It is motivated by the fact that in time series prediction the accuracy of the target variable depends not only on the variable itself but also on exogenous variables. Our algorithm has been developed using the Scala programming language for the Apache Spark open source software.

We firstly conducted theoretical analysis in order to study the correlation thresholds among the variables to find when the multivariate approach is useful. Five different correlations were considered, 0, 0.25, 0.5, 0.75, 1. The results showed that if an exogenous time series does not exhibit a significant correlation with the future values of the target variable, the accuracy will



(a) Time of prediction of the different models being the size of the dataset 1

(b) Time of prediction with different sizes of the datasets and different numbers of machines in the cluster(MV-kWNN)

Figure 13: Runtimes analysis of predictions

not be improved but the error will increase only slightly. This demonstrates that our algorithm is quite robust in terms of accuracy even if the exogenous variables are not properly selected. By contrast, if the exogenous variable shows greater correlation with the future values of the target variable than the target variable itself, the error decreases sharply.

We then present a successful case study of a real-world problem, using two data sources from the Spanish electricity market, electricity demand and electricity prices. We compare the performance of MV-kWNN with other standard multivariate forecasting algorithms, such as ARX, ARMAX and ARIMAX and with other nonlinear multivariate forecasting algorithms, such as neural networks and random forests. Our results show that MV-kWNN outperforms the other algorithms. Statistical tests have been applied to prove meaningful statistical differences among all the considered methods.

Finally, a scalability analysis is conducted with up to 200 times the size of the dataset showing a linear relation between data size and execution times and, hence, confirming the suitability of the MV-kWNN algorithm for applications to big data.

7. Appendix

V1	0.6032	0.6360	0.6126	0.5748	0.5284	0.4925	0.4653	0.4539	0.4338	0.4240	0.4031	0.4301	0.4682	0.4877	0.5114	0.5257	?	?
V2	0.3535	0.2321	0.3351	0.3310	0.2288	0.1850	0.1672	0.1548	0.1394	0.1103	0.0690	0.0690	0.1517	0.1793	0.1850	0.1793	?	?
V3	0.4523	0.5195	0.4936	0.5213	0.4309	0.4052	0.5832	0.5683	0.5046	0.5911	0.2435	0.5863	0.3609	0.5974	0.5720	0.3042	?	?

w=4 h=2

(a) Selecting the past w values of three time series to predict h values of each one.

w Pattern
(0.4682,0.4877,0.5114,0.5257)(0.1517,0.1793,0.185,0.1793)(0.3609,0.5974,0.572,0.3042)

w's	h's	dist
...	(0.6032,0.636)(0.3535,0.2321)(0.4523,0.5195)	NA
(... ,... , 0.6032,0.636)(... ,... , 0.3535,0.2321)(... ,... , 0.4523,0.5195)	(0.6126,0.5748)(0.3351,0.331)(0.4936,0.5213)	NA
(0.6032,0.636,0.6126,0.5748)(0.3535,0.2321,0.3351,0.331)(0.4523,0.5195,0.4936,0.5213)	(0.5284,0.4925)(0.2288,0.185)(0.4309,0.4052)	0.2299
(0.6126,0.5748,0.5284,0.4925)(0.3351,0.331,0.2288,0.185)(0.4936,0.5213,0.4309,0.4052)	(0.4653,0.4539)(0.1672,0.1548)(0.5832,0.5683)	0.1727
(0.5284,0.4925,0.4653,0.4539)(0.2288,0.185,0.1672,0.1548)(0.4309,0.4052,0.5832,0.5683)	(0.4338,0.424)(0.1394,0.1103)(0.5046,0.5911)	0.1045
(0.4653,0.4539,0.4338,0.424)(0.1672,0.1548,0.1394,0.1103)(0.5832,0.5683,0.5046,0.5911)	(0.4031,0.4301)(0.069,0.069)(0.2435,0.5863)	0.1323
(0.4338,0.424,0.4031,0.4301)(0.1394,0.1103,0.069,0.069)(0.5046,0.5911,0.2435,0.5863)	(0.4682,0.4877)(0.1517,0.1793)(0.3609,0.5974)	0.1616
(0.4031,0.4301,0.4682,0.4877)(0.069,0.069,0.1517,0.1793)(0.2435,0.5863,0.3609,0.5974)	(0.5114,0.5257)(0.185,0.1793)(0.572,0.3042)	0.1042

k=2

(b) Grouping the values of the time series in windows of w and h values and selecting the closest neighbours.

h's (k closest neighbours) * weights	dist (k closest neigbours)	weights (1/dist^2)
(39.6979,38.8011)(12.7568,10.0938)(46.1769,54.0927)	0.1045	91.5120
(47.0642,48.3802)(17.0256,16.501)(52.6412,27.9956)	0.1042	92.0301

∑ h's (k closest neighbours) * weights	∑ weights
(86.7621,87.1813)(29.7824,26.5948)(98.8181,82.0883)	183.5421

Predictions ((∑ h's * weights) / (∑ weights))		
(0.4727,0.475)	(0.1623,0.1449)	(0.5384,0.4472)
V1	V2	V3

(c) Calculating the predictions of the three time series.

Figure 14: A numerical example showing how MV-kWNN works

Acknowledgments.

The authors would like to thank the Spanish Ministry of Economy and Competitiveness and Junta de Andalucía for their support under projects TIN2014-55894-C2-R and P12-TIC-1728, respectively. Additionally, the authors wish to express their gratitude to the T-Systems Iberia company since all experiments were carried out on its Open Telekom Cloud Platform based on the Open-Stack open source.

References

[1] G. Asencio-Cortés, E. Florido, A. Troncoso, and F. Martínez-Álvarez. A novel methodology to predict urban traffic congestion with ensemble learning. *Soft Computing*, 20(11):4205–4216, Nov 2016.

[2] A. Azadeh, S.F. Ghaderi, and S. Sohrabkhani. Annual electricity consumption forecasting by neural network in high energy consuming industrial sectors. *Energy Conversion and Management*, 49(8):2272 – 2278, 2008.

[3] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis*. Wiley, 2013.

[4] M. Brown, C. Barrington-Leigh, and Z. Brown. Kernel regression for real-time building energy analysis. *Journal of Building Performance Simulation*, 5(4):263–276, 2012.

[5] P. Čech, J. Kohout, J. Lokoč, T. Komárek, J. Maroušek, and T. Pevný. Feature extraction and malware detection on large https data using mapreduce. In *Proceedings of the 9th International Conference on Similarity Search and Applications (SISAP)*, pages 311–324, 2016.

[6] T. Colombo, I. Koprinska, and M. Panella. Maximum length weighted nearest neighbor approach for electricity load forecasting. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.

[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] A. M. Fernández, J. F. Torres, A. Troncoso, and F. Martínez-Álvarez. Automated spark clusters deployment for big data with standalone applications integration. *Lecture Notes in Artificial Intelligence*, 9868:150–159, 2016.

[9] E. Florido, F. Martínez-Álvarez, A. Morales-Esteban, J. Reyes, and J.L. Aznarte-Mellado. Detecting precursory patterns to enhance earthquake prediction in chile. *Computers & Geosciences*, 76(Supplement C):112 – 120, 2015.

[10] A. Galicia, J. F. Torres, F. Martínez-Álvarez, and A. Troncoso. Scalable forecasting techniques applied to big electricity time series. In *Proceedings of the 14th International Work-Conference on Artificial Neural Networks (IWANN)*, pages 165–175, 2017.

[11] S. García, A. Fernández, J. Luengo, and F. Herrera. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Computing*, 13(10):959, Dec 2008.

[12] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. *ACM SIGOPS Operating System Review*, 37(5):29–43, 2003.

[13] H. S. Hippert, C. E. Pedreira, and R. C. Souza. Neural networks for short-term load forecasting: a review and evaluation. *IEEE Transactions on Power Systems*, 16(1):44–55, Feb 2001.

[14] E. Huvio, J. Grönvall, and K. Främling. Tracking and tracing parcels using a distributed computing approach. In *Proceedings of the 14th Annual conference for Nordic researchers in logistics (NOFOMA'2002), Trondheim, Norway*, pages 29–43, 2002.

[15] H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. *The Annals of Applied Statistics*, 2(3):841–860, 2008.

[16] Y. Kusunose and R. Mahmood. Imperfect forecasts and decision making in agriculture. *Agricultural Systems*, 146(Supplement C):103 – 110, 2016.

[17] J. Lee, S. Lee, J. Kim, D. Song, and H. Jeong. A middleware platform for the validation and utilization of short-term weather forecast data for

office buildings. *Energy and Buildings*, 149(Supplement C):192 – 203, 2017.

[18] H. Li, S. Guo, C. Li, and J. Sun. A hybrid annual power load forecasting model based on generalized regression neural network with fruit fly optimization algorithm. *Knowledge-Based Systems*, 37:378 – 387, 2013.

[19] T. Liu, C. Rosenberg, and H. A. Rowley. Clustering billions of images with large scale nearest neighbor search. In *IEEE Workshop on Applications of Computer Vision (WACV)*, pages 28–28, 2007.

[20] H. Lütkepohl. *Introduction to multiple time series analysis*. Springer, 1991.

[21] L. Macías-García, J. M. Luna-Romera, J. García-Gutiérrez, M. Martínez-Ballesteros, J. C. Riquelme-Santos, and R. González-Cámpora. A study of the suitability of autoencoders for preprocessing data in breast cancer experimentation. *Journal of Biomedical Informatics*, 72:33 – 44, 2017.

[22] J. Maillo, S. Ramírez, I. Triguero, and F. Herrera. knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data. *Knowledge-Based Systems*, 117:3 – 15, 2017.

[23] F. Martínez-Álvarez, A. Troncoso, G. Asencio-Cortés, and J. C. Riquelme. A survey on data mining techniques applied to energy time series forecasting. *Energies*, 8:1–32, 2015.

[24] F. Martínez-Álvarez, A. Troncoso, J. C. Riquelme, and J. S. Aguilar Ruiz. Energy time series forecasting based on pattern sequence similarity. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1230–1243, 2011.

[25] A. H. Murphy. What is a good forecast? an essay on the nature of goodness in weather forecasting. *Weather and Forecasting*, 8(2):281–293, 1993.

[26] N. Nodarakis, A. Rapti, S. Sioutas, A. K. Tsakalidis, D. Tsolis, G. Tzimas, and Y. Panagis. (a)knn query processing on the cloud: A survey. In *Second International Workshop on Algorithmic Aspects of Cloud Computing (ALGOCLOUD)*, pages 26–40, 2017.

[27] B. O'Connor, R. Balasubramanyan, B. Routledge, and N. Smith. From tweets to polls: Linking text sentiment to public opinion time series, 2010.

[28] R package (neuralnet). On-line. *https://CRAN.R-project.org/package=neuralnet*, 2016.

[29] R package (randomForestSRC). On-line. *https://CRAN.R-project.org/package=randomForestSRC*, 2017.

[30] J. A. Parejo, J. García, A. Ruiz-Cortés, and J. C. Riquelme. Statservice: Herramienta de análisis estadístico como soporte para la investigación con metaheurísticas. In *Actas del VIII Congreso Expañol sobre Metaheurísticas, Algoritmos Evolutivos y Bio-inspirados*, 2012.

[31] R. Pérez-Chacón, R. L. Talavera-Llames, F. Martínez-Álvarez, and A. Troncoso. Finding electric energy consumption patterns in big time series data. In *Proceedings of the 13th International Conference on Distributed Computing and Artificial Intelligence*, pages 231–238, 2016.

[32] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhua, J. Zakaria, and E. Keogh. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Trans. Knowl. Discov. Data*, 7(3):10:1–10:31, September 2013.

[33] M. Rana, I. Koprinska, and V. G. Agelidis. Univariate and multivariate methods for very short-term solar photovoltaic power forecasting. *Energy Conversion and Management*, 121(Supplement C):380 – 390, 2016.

[34] M. Rana, I. Koprinska, and A. Troncoso. Forecasting hourly electricity load profile using neural networks. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 824–831, 2014.

[35] S. Russell and P. Norvig. *Artificial Intelligence: A modern Approach (3rd Ed.)*. Pearson Prentice Hall, 2011.

[36] N. D. Savio and K. Nikolopoulos. A strategic forecasting framework for governmental decision-making and planning. *International Journal of Forecasting*, 29(2):311 – 321, 2013.

[37] K. Sun, H. Kang, and H. Park. Tagging and classifying facial images in cloud environments based on knn using mapreduce. *Optik - International Journal for Light and Electron Optics*, 126(21):3227 – 3233, 2015.

[38] R. L. Talavera-Llames, R. Pérez-Chacón, M. Martínez-Ballesteros, A. Troncoso, and F. Martínez-Álvarez. *A Nearest Neighbours-Based Algorithm for Big Time Series Data Forecasting*, pages 174–185. Springer International Publishing, Cham, 2016.

[39] C. Thirumalai, K. S. Sree, and H. Gannu. Analysis of cost estimation function for facebook web click data. In *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, volume 2, pages 172–175, April 2017.

[40] J. F. Torres, A. M. Fernández, A. Troncoso, and F. Martínez-Álvarez. Deep learning-based approach for time series forecasting with application to electricity load. In *Proceedings of the International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC)*, pages 203–212, 2017.

[41] A. Troncoso, J. M. Riquelme-Santos, A. G. Expósito, J. L. Martínez-Ramos, and J. C. Riquelme. Electricity market price forecasting based on weighted nearest neighbors techniques. *IEEE Transactions on Power Systems*, 22(3):1294–1301, 2007.

[42] Y. Wang and G. Li. An efficient data aggregation scheme in wireless sensor networks. In Y. Wang and X. Zhang, editors, *Internet of Things*, pages 25–32, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[43] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, O. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[44] F. Xia, L. T. Yang, L. Wang, and A. Vinel. Internet of things. *International Journal of Communication Systems*, 25(9):1101, 2012.

[45] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets:

A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 2–2, 2012.

[46] M. Zekic-Susac, A. Sarlija, A. Has, and A. Bilandzic. Predicting company growth using logistic regression and neural networks. *Croatian Operational Research Review*, 149(2):229 – 248, 2016.

6.4. Finding electric energy consumption patterns in big time series data

El artículo en congreso internacional publicado es el siguiente:

- R. Pérez-Chacón, R. L. Talavera-Llames, F. Martínez-Álvarez and A. Troncoso. Finding electric energy consumption patterns in big time series data. In *Proceedings of the 13th International Conference on Distributed Computing and Artificial Intelligence*, pages 231-238, 2016.

Finding electric energy consumption patterns in big time series data

R. Pérez-Chacón, R. L. Talavera-Llames, F. Martínez-Álvarez, and A. Troncoso

Division of Computer Science, Universidad Pablo de Olavide, ES-41013 Seville, Spain
{rpercha,rltallla,fmaralv,ali}@upo.es

Abstract. In recent years the available volume of information has grown considerably due to the development of new technologies such as the sensor networks or smart meters, and therefore, new algorithms able to deal with big data are necessary. In this work the distributed version of the k-means algorithm in the Apache Spark framework is proposed in order to find patterns from a big time series. Results corresponding to the electricity consumptions for years 2011, 2012 and 2013 for two buildings from a public university are presented and discussed. Finally, the performance of the proposed methodology in relation to the computational time is compared with that of Weka as benchmarking.

Keywords: Big data, time series, patterns, clustering.

1 Introduction

Rapid and huge data storage is in frequent use nowadays. This new scenario causes extreme difficulties to both efficiently process and store such big amount of data [6]. In this context, much effort is being devoted to enhance existing data mining techniques in order to process, manage and discover knowledge from this big data [13].

The limitations of the MapReduce paradigm [3] for iterative algorithms development have led to new paradigms, such as Apache Spark [8], which is an open source software project. Among its most important capacities, multi-pass computations, high-level operators, diverse languages usage, in addition to its own language called Scala, are most notable. Moreover, a machine learning library, MLlib [7], is also integrated within the framework.

The objective of this work is the discovery of patterns in big time series of electricity consumption. Given its size, the collected data cannot be processed with classical machine learning approaches. Therefore, implementations for distributed computing must be used and, in particular, a distributed methodology based on the, still relatively unknown, parallelized version of k-means++ is proposed. This methodology has been developed by using MLlib in the framework Apache Spark, under the Scala programming language. Real–world big data sets collected from a sensor network located in several buildings of Pablo de Olavide

University have been analyzed. The successful analysis of these patterns is expected to be used for efficient management of the university electricity resources, as well as for characterizing the electricity consumption over time.

Increased attention has been paid to big data clustering in recent years. A survey on this topic can be found in [5]. Specifically, several approaches have been recently proposed to apply clustering to big time series data. Namely, in [4] the authors propose a new clustering algorithm based on a previous clustering of a sample of the input data. The dynamic time warping was tested to measure the similarity between big time series in [14]. In [16] a data processing based on mapreduce was used to obtain clusters. A distributed method for the initialization of the k-means is proposed in [2]. However, there is still much research to be conducted in this field, especially considering that very few works have been published.

The study of electricity profiles by means of clustering techniques for small and medium datasets has been studied in the literature. In [15] a methodology based on the visualization to obtain the clusters is provided. In [12] the authors examined Spanish electricity prices, discovering some associated patterns to different days and to different seasons. The study was performed by applying crisp clustering, in contrast to the study carried out in [11], where fuzzy clustering was also shown to be useful in this context.

Clustering consumption data was also the goal in [10] but, this time, the authors went one step further and used this information as input for consumption forecasting.

Later in 2012, classification and clustering of electricity demand patterns in industrial parks was addressed [9]. In this work, a data processing system to analyze energy consumption patterns in Spanish parks, based on the cascade application of a Self-Organizing Maps and the k-means algorithm, was introduced.

As for the particular case of clustering big time series consumption data, there is no study carried out so far, to the best of the authors' knowledge. And this is precisely the reason why this study is presented.

The remainder of the paper is structured as follows. In Section 2 the proposed method to discover patterns in time series is described. Section 3 presents the experimental results corresponding to the clustering of the energy consumption coming from a sensor network of building facilities. Finally, Section 4 summarizes the main conclusions drawn from this study.

2 Methodology

This section describes the methodology proposed in order to find consumption patterns in electricity-related big time series data. In particular, the k-means algorithm, included in MLlib, is used in a Spark context to obtain clusters that define consumption patterns.

Figure 1 shows the key steps of the proposed methodology to obtain patterns as a result of the clustering. The first phase consists of data cleansing and the

transformations carried out over a RDD variable of Spark, in order to use it in a distributed way. The dataset is the electricity consumption time series from two buildings from a public university for every fifteen minutes of the years 2011, 2012 and 2013. Each row of the dataset contains the following information: building name, date (split into five fields) and the electricity consumption data. Nevertheless, some of these rows contains accumulated consumption power data due to existing missing values, which were successfully preprocessed to learn correctly the models in the next phase. The preprocess stage is properly detailed in the Section 3.

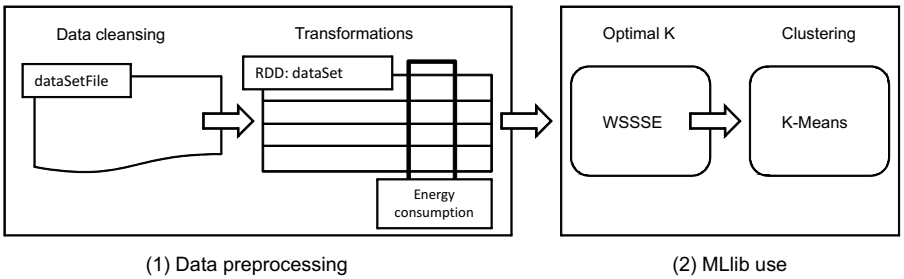


Fig. 1. Illustration of the proposed methodology.

Once the RDD dataset is created, it is necessary to group it in rows of 96 values (4 values per hour per 24 hours of a day) and reduce the dimension of the original RDD dataset before creating a model. This reduction consists in removing the building name field, and transforming the date into a numerical index. Thus, each row finally contains the 96 values corresponding to electricity consumption for a given day.

The second phase consists in the use of MLlib. Firstly, it is necessary to obtain an optimal number of clusters k, which will be used as an input parameter in the k-means algorithm. For that, the Within Set Sum of Squared Errors (WSSSE) index, defined by the sum of the squared Euclidean distance between the elements of a cluster and its centroid for all clusters and instances of the big data, is computed when applying the k-means for a certain number of clusters. In fact, the optimal k is usually the one which is a local minimum in the WSSSE graph.

MLlib includes a parallelized version of k-means++ [1], called k-means||, that it is used in this work to obtain the resulting models. The k-means|| algorithm runs the k-means algorithm a number of times in a concurrent way, returning the best clustering result.

In Figure 2 one execution of the the parallelized k-means algorithm version is described. Firstly, the RDD dataset is parallelized in n nodes for each concurrent run of the k-means. Therefore, n provisional centroids are obtained. Secondly, Spark shuffles the n centroids to provide a resulting centroid for each concurrent run. Finally, once all concurrent runs have been executed, the k-means||

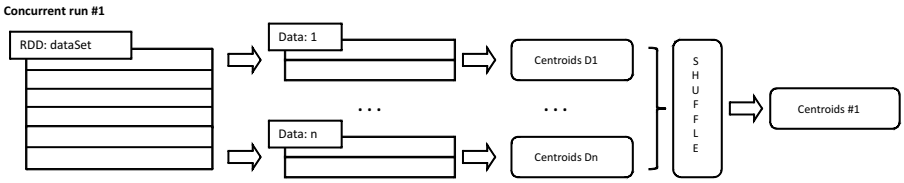


Fig. 2. One concurrent execution of the parallelized k-means algorithm.

algorithm computes the WSSSE for each centroid (note that there are as many centroids as number of concurrent runs). Thus, the algorithm returns the centroid that minimizes the WSSSE as the best centroid.

3 Results

The datasets used are related to the electrical energy consumption in two buildings located at a public university for years 2011, 2012 and 2013. The consumption is measured every fifteen minutes during this period. This makes a total of 35040 instances for years 2011 and 2013, and 35136 for the year 2012.

Note that there were several missing values ($< 3\%$). However, subsequent time stamps store the accumulated consumption for such instances. Therefore, the cleansing process consisted in searching for such values and assuming that consumption had been constant during these periods of time. That is, the stored value after missing values is divided by the number of consecutive registered missing values and assigned to each one.

Figure 3 shows the error obtained when applying the k-means for a number of clusters varying from 2 to 15 for the consumption of electricity of the years 2011, 2012 and 2013. The error used was the sum of squares of the distance between the points of each cluster. The error decreases smoothly for values greater than 6, do to this a number of clusters equal to 6 can be selected to provide satisfactory results.

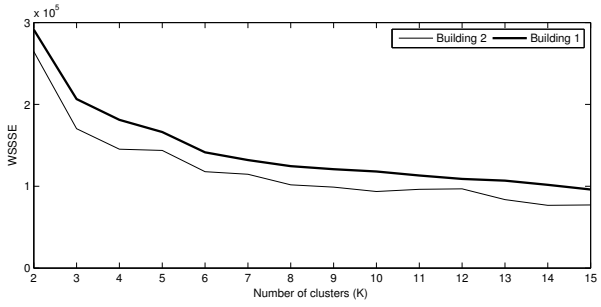


Fig. 3. Errors versus number of clusters for each building.

Figure 4 presents the classification into 6 clusters obtained for K-means for the year 2013 (similar figures were obtained for the years 2011 and 2012). With just a quick look, the weekends, the working days and the typical periods of vacations in the university such as the Easter week (values from 83 to 90), the summer holidays (values from 213 to 243) or Christmas (values from 356 to 365) can be clearly differentiated.

The percentage of days classified into 6 clusters for each building is shown in Table 1. The last row presents the average of the electricity consumption for all the days associated with the cluster. Although it seems that the working days are equally distributed, a detailed analysis from Table 1 and Figure 4 reveals interesting patterns related to temperature and days with or with no scheduled teaching. For Building 1, teaching days with low and high temperatures (winter and summer) belong to the clusters of greater consumption, that is, cluster 4 and 5 respectively, and finally, teaching days with no extreme temperatures (autumn) are classified into cluster 2 of moderate consumption. Similar patterns can be found for Building 2.

Characteristic curves of each cluster are depicted in Figure 5. It can be observed that clusters 1 and 2 for Building 1, and clusters 1 and 2 for Building 2 are clusters composed of days of low consumption, namely weekends and holidays. Likewise, the remaining clusters correspond to working days, which are days of greater consumption.

Table 1. Percentage of days for each cluster in years 2011, 2012 and 2013.

Days	Building 1					
	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Monday	5.73%	15.29%	12.74%	17.83%	10.83%	37.58%
Tuesday	5.73%	19.75%	12.10%	19.11%	8.92%	34.39%
Wednesday	5.77%	20.51%	12.18%	16.67%	8.33%	36.54%
Thursday	6.41%	16.67%	15.38%	17.95%	8.33%	35.26%
Friday	10.90%	11.54%	14.74%	14.74%	10.26%	37.82%
Saturday	42.68%	2.55%	48.41%	0.00%	0.64%	5.73%
Sunday	16.56%	1.27%	81.53%	0.64%	0.00%	0.00%
Average (in kW)	1.90	3.37	4.57	5.47	6.54	6.94

Days	Building 2					
	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Monday	14.65%	38.22%	14.01%	16.56%	10.83%	5.73%
Tuesday	15.92%	32.48%	15.92%	19.11%	8.28%	8.28%
Wednesday	16.03%	35.90%	14.74%	15.38%	7.69%	10.26%
Thursday	19.87%	35.90%	15.38%	12.82%	7.69%	8.33%
Friday	19.23%	41.67%	13.46%	12.18%	8.97%	4.49%
Saturday	80.89%	17.83%	0.00%	0.64%	0.00%	0.64%
Sunday	96.82%	1.91%	0.00%	0.64%	0.00%	0.64%
Average (in kW)	1.41	2.39	3.96	4.71	5.43	6.39

Figure 6 shows the relation between the CPU time and the size of the dataset for k-means from Weka and Spark. This size has been set to 370 years, by syn-

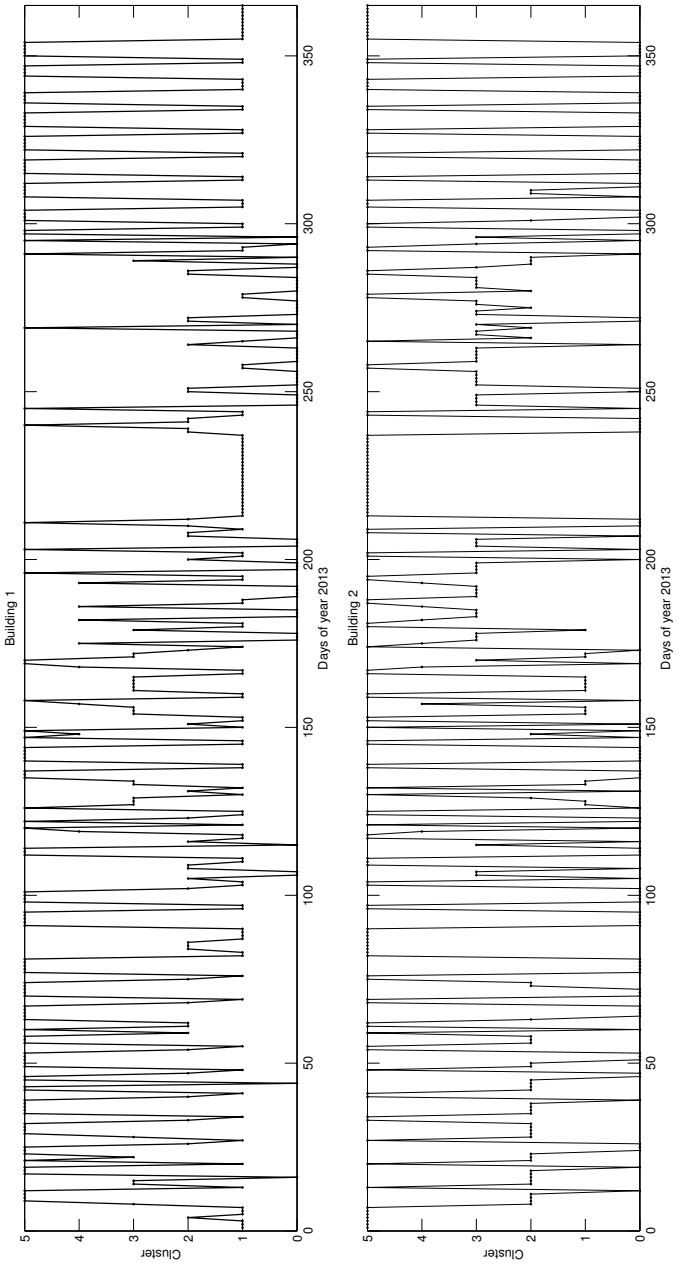


Fig. 4. Classification of the electricity consumption for each building.

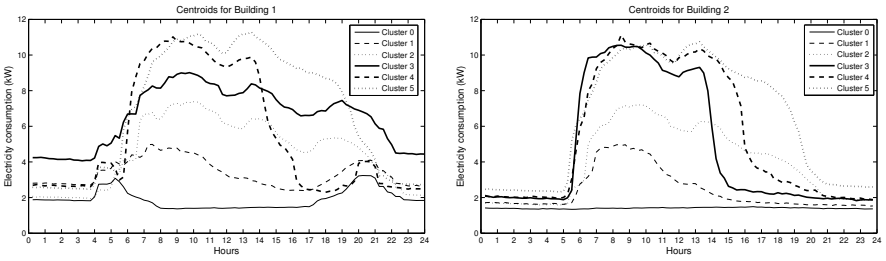


Fig. 5. Centroids for each building in years 2011, 2012 and 2013.

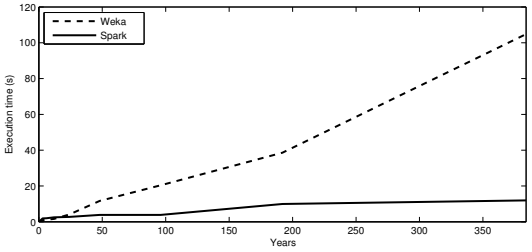


Fig. 6. CPU time for different sizes of datasets when using Weka and Spark.

thetically generating such years. As a consequence of the results, it can be noticed that Weka has an exponential growth when the number of years comprising the time series increases, being remarkable the differences with Spark when the dataset is considered as big data.

4 Conclusions

In this work, a real big time series data composed of electricity consumptions has been analyzed by means of the k-means algorithm distributed version for Apache Spark. This parallelized version of the algorithm allows the discovery of daily consumption behaviors with a low computational cost. The results show different kinds of days according to the daily consumption as well as the identification of significant patterns related to working days with or with no scheduled teaching. Moreover, the CPU time of the proposed methodology has been compared to Weka, as a reference tool in data mining, proving to be a good solution for the big data clustering. Future works will be directed in the prediction of big time series once known the previous clustering, and the discovery of patterns for the classification of all the buildings of an organization in the context of smart cities.

Acknowledgments.

The authors would like to thank the Spanish Ministry of Economy and Competitiveness, Junta de Andalucía for the support under projects TIN2014-55894-C2-R and P12-TIC-1728, respectively.

References

1. B. Bahmani, A. Moseley, R. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.
2. M. Capó, A. Pérez, and J. A. Lozano. A Recursive k-means Initialization Algorithm for Massive Data. In *Proceedings of the Spanish Association for Artificial Intelligence*, 2015.
3. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
4. R. Ding, Q. Wang, Y. Dan, Q. Fu, H. Zhang, and D. Zhang. Yading: fast clustering of large-scale time series data. *Proceedings of the VLDB Endowment*, 8(5):473–484, 2015.
5. A. Fahad, N. Alshatri, Z. Tari, A. Alamri, A. Y. Zomaya, I. Khalil, F. Sebti, and A. Bouras. A survey of clustering algorithms for big data: Taxonomy & empirical analysis. *IEEE Transactions on Emerging Topics in Computing*, 5:267–279, 2014.
6. A. Fernández, S. del Río, V. López, A. Bawakid, M. J. del Jesús, J. M. Benítez, and F. Herrera. Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. *WIREs Data Mining and Knowledge Discovery*, 4(5):380–409, 2014.
7. Machine Learning Library (MLlib) for Spark. On-line. *http://spark.apache.org/docs/latest/mllib-guide.html*, 2015.
8. M. Hamstra, H. Karau, M. Zaharia, A. Knwinski, and P. Wendell. *Learning Spark: Lightning-Fast Big Analytics*. O' Really Media, 2015.
9. L. Hernández, C. Baladrón, J. M. Aguiar, B. Carro, and A. Sánchez-Esguevillas. Classification and clustering of electricity demand patterns in industrial parks. *Energies*, 5:5215–5228, 2012.
10. H. R. S. Keyno, F. Ghaderi, A. Azade, and J. Razmi. Forecasting electricity consumption by clustering data in order to decline the periodic variable's affects and simplification the pattern. *Energy Conversion and Management*, 50(3):829–836, 2009.
11. F. Martínez-Álvarez, A. Troncoso, J. C. Riquelme, and J. M. Riquelme. Discovering patterns in electricity price using clustering techniques. In *Proceedings of the International Conference on Renewable Energy and Power Quality*, pages 245–252, 2007.
12. F. Martínez-Álvarez, A. Troncoso, J. C. Riquelme, and J. M. Riquelme. Partitioning-clustering techniques applied to the electricity price time series. *Lecture Notes in Computer Science*, 4881:990–991, 2007.
13. M. Minelli, M. Chambers, and A. Dhiraj. *Big Data, Big Analytics: emerging business intelligence and analytics trends for today's businesses*. John Wiley and Sons, 2013.
14. T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Transactions on Knowledge Discovery from Data*, 7(3):267–279, 2014.
15. J. J. Van Wijk and E. R Van Selow. Cluster and calendar based visualization of time series data. In *Proceedings of the International IEEE Symposium on Information Visualization*, 1999.
16. W. Zhao, H. Ma, and Q. He. Parallel k-means clustering based on mapreduce. *Lecture Notes in Computer Science*, 5391:674–679, 2009.

6.5. Multi-step forecasting for big data time series based on ensemble learning

El artículo publicado en revista indexada es el siguiente:

- A. Galicia, R. Talavera-Llames, A. Troncoso, and I. Koprinska, and F. Martínez-Álvarez. Multi-step forecasting for big data time series based on ensemble learning. *Knowledge-Based Systems*, 163:830-841, 2019.
    - Estado: publicado (disponible on-line)
    - Índice de impacto (JCR 2017): 4.396
    - Área de conocimiento:
        - Computer Science, Artificial Intelligence. Ranking 14 / 132 - Q1

Knowledge-Based Systems 163 (2019) 830–841

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Multi-step forecasting for big data time series based on ensemble learning

A. Galicia [a], R. Talavera-Llames [a], A. Troncoso [a,*], I. Koprinska [b], F. Martínez-Álvarez [a]

[a] Data Science & Big Data Lab, Universidad Pablo de Olavide, ES-41013 Seville, Spain
[b] School of Information Technologies, University of Sydney, Sydney, NSW, Australia

ARTICLE INFO

Article history:
Received 15 March 2018
Received in revised form 4 October 2018
Accepted 5 October 2018
Available online 12 October 2018

Keywords:
Big data
Ensemble
Electricity time series
Forecasting

ABSTRACT

This paper presents ensemble models for forecasting big data time series. An ensemble composed of three methods (decision tree, gradient boosted trees and random forest) is proposed due to the good results these methods have achieved in previous big data applications. The weights of the ensemble are computed by a weighted least square method. Two strategies related to the weight update are considered, leading to a static or dynamic ensemble model. The predictions for each ensemble member are obtained by dividing the forecasting problem into h forecasting sub-problems, one for each value of the prediction horizon. These sub-problems have been solved using machine learning algorithms from the big data engine Apache Spark, ensuring the scalability of our methodology. The performance of the proposed ensemble models is evaluated on Spanish electricity consumption data for 10 years measured with a 10-minute frequency. The results showed that both the dynamic and static ensembles performed well, outperforming the individual ensemble members they combine. The dynamic ensemble was the most accurate model achieving a MRE of 2%, which is a very promising result for the prediction of big time series. Proposed ensembles are also evaluated using solar power from Australia for two years measured with 30-min frequency. The results are successfully compared with Artificial Neural Network, Pattern Sequence-based Forecasting and Deep Learning, improving their results.

1. Introduction

Advances in technology have led to an increasing generation and storage of massive data in recent years [1,2]. These data need to be efficiently processed in order to extract useful and valuable knowledge. Thus, the development of new tools for dealing with big data has become a critical issue. An essential component of the nature of big data is that information is usually captured over time at different points, resulting in big data time series [3]. This information can be analysed for various purposes: to predict the future values, to establish relations among variables, to detect anomalous values, or to discover patterns.

The main existing frameworks for the massive data processing have been developed thanks to leading technology companies. For example, the MapReduce technology was developed by Google [4]; it divides the input data into small blocks, processes them and then aggregates the output into a single solution. Based on this paradigm, Yahoo! developed an open-source implementation called Hadoop [5] and later Spark was developed by the University of Berkeley in California [6].

Spark maximizes parallelization of data processing in-memory, achieving much faster processing speed than Hadoop. Spark also has specific modules for mining big data, such as the Apache Spark's Machine Learning Library (MLlib) [7]. Although its native language is Scala, it also supports Python, R and Java. Spark is seen as a standard framework for data-intensive computing, and is being used in a wide range of problems such as climate data [8], water system [9], earthquakes [10], entity matching for information integration and data cleansing [11] or energy data in buildings [12]. In addition, studies of the performance of the Spark system are shown in [13] and [14].

This study is framed in the time series forecasting context, with arbitrary time horizon and in a big data environment. The previous work in [15] assessed the performance of MLlib for the forecasting of big data time series. A set of scalable algorithms was studied and adapted for very large time series forecasting. In particular, representative methods of different nature, such as linear regression, decision trees, gradient boosted trees and random forest were analysed. The results reported were promising and, for this reason, we now explore the suitability of combining some of these algorithms into ensembles to forecast big data time series.

* Corresponding author.
E-mail addresses: agalde@alu.upo.es (A. Galicia), rltallla@upo.es (R. Talavera-Llames), atrolor@upo.es (A. Troncoso), irena.koprinska@sydney.edu.au (I. Koprinska), fmaralv@upo.es (F. Martínez-Álvarez).

https://doi.org/10.1016/j.knosys.2018.10.009
0950-7051/

In particular, three of the aforementioned methods (decision trees, gradient boosted trees and random forest) have been used to develop a novel ensemble forecasting algorithm. Linear regression has been discarded because its performance, although acceptable in general terms, was too poor when compared to the other methods. The ensemble approach assigns different weights to every method by using a weighted square least method, which optimizes the contribution of each individual forecast in the combined forecast for a given forecasting time. Therefore, our ensemble is not a typical boosting ensemble, but a weighed voting ensemble as we combine three base models by using a weighed majority vote, where the weights are calculated based on the previous performance of these models using a least squares method. We propose two different strategies for training the ensemble models: static, in which the training set remains the same for each target sample to be forecasted, and dynamic, in which the training set slides forwards and changes for every target sample to be forecasted. The performance of the proposed ensemble algorithm has been evaluated using electricity consumption data from Spain. The ensemble outperforms all three methods it combines when they are used individually. In particular, the dynamic ensemble was the best performing model achieving mean relative errors of about 2%, which is a very competitive results [16].

An important feature of the proposed approach is its ability to deal with prediction horizons of arbitrary length. In this sense, if the prediction horizon is composed of h samples, h independent models are generated and simultaneously processed.

Although we used the MLlib implementations of the three base prediction methods, there were some limitations that we had to overcome. On one hand, the regression techniques available in MLlib do not support more than one step ahead prediction. On the other hand, the RDD (Resilient Distributed Dataset) data structure used by Spark is not designed to guarantee the order of data, which is a critical aspect in time series processing . The handling of these two key limitations is another contribution of this work.

In summary, the main contributions of this work are:

(1) We propose a weighted voting ensemble that uses a least squares method to calculate the weights of the base models.

(2) We develop two versions of this ensemble, static and dynamic.

(3) We show how this ensemble can be evaluated on big data for multi-step ahead forecasting by decomposing the task into multiple prediction problems, which can be solved by the Apache Spark big data engine.

(4) We conduct a comprehensive evaluation using Spanish electricity data for 10 years, measured at 10-min intervals, demonstrating that both ensemble members performed, outperforming the base models they combine, and particularly showing the potential of dynamic ensembles for big data forecasting.

(5) Solar photovoltaic dataset from Australia has been used to compare proposed ensembles with algorithms of different nature, such us deep learning, pattern sequence-based forecasting and artificial neural networks.

Section 2 reviews the literature related to time series forecasting techniques, machine learning for big data and ensemble learning. A theoretical background is included in Section 3, where the multi-step methodology is proposed. Section 4 presents and discusses the results of Spanish electricity data. Section 5 discusses and compares the results of Australian solar data . Finally, Section 6 summarizes the main conclusions.

2. Related work

This section discusses the most relevant related works, focusing on big data. Although short and medium term time series forecasting have been extensively studied in the literature, there are very few works on time series forecasting for big data.

In general, the methods for predicting time series can be classified into classical methods based on Box and Jenkins [17], such as ARIMA and GARCH; and data mining methods, such as Support Vector Machine (SVM), k Nearest Neighbour techniques (kNN) and Artificial Neural Networks (ANN). For a taxonomy of these techniques applied to energy time series forecasting, the reader is referred to [16].

However, due to the high computational cost, the majority of the data mining techniques cannot be applied when big data have to be processed. Therefore, big data mining techniques [18,19] are being developed for distributed computing in order to solve typical tasks as classification, clustering or regression. A brief description of the main advances in this area is given below.

Several MapReduce-based approaches for big data scenarios have been recently provided for classification tasks. The SVM algorithm was recently adapted to the field of high performance computing giving rise to parallel SVMs [20]. Based on Spark, several parallel implementations of the kNN algorithm have been proposed in [21–23]. Also, a MapReduce-based framework focused on instance reduction methods was proposed in [24] to reduce the computational cost and storage requirements of kNN. Deep learning has been also used for industry process planning in [25].

In recent years, increased attention has been paid to big data clustering. A survey on this topic can be found in [26,27]. In the particular field of big data time series, K-means has been successfully applied in [28]. Likewise, the challenging task of determining the optimal number of partitions has been addressed in [29], where scalable approaches to determine the quality of the generated partitions using clustering were proposed.

Very few works have been published on using big data for regression tasks, hence there is much room for improvement. A survey on big data forecasting is presented in [30]. Some regression algorithms based on cloud and big data technologies have been used for earthquake prediction in California [31]. An approach based on kNN to forecast big data time series was introduced in 2016 in [22] and approaches based on deep learning have also been published in 2017 [32]. A scalable fuzzy system for regression is proposed in [33], as the performance of the fuzzy rules depends on the size of the problem.

A variety of ensemble methods have been proposed and successfully used in practical applications [34]. The field of ensembles was developed to improve the accuracy of an automated decision-making system, with the aim of reducing variance. Since then, ensembles have been widely used in different machine learning problems, for prediction and classification, feature selection, missing feature, incremental learning, confidence estimation, error correction, among others.

Polikar [35] provided an overview of ensembles, their properties and how they can be applied to different tasks. Ensemble learning is being used for streaming analysis, a survey can be found in [36]. Ensemble techniques based on threes are the most recurrent topic in the literature for big data. This is mainly due to the easy adaptation of these algorithms for distributed computing. Random Forest has been applied to some specific problems, showing good performance for large datasets [37]. Analogously, regression trees have been constructed using parallel learning with MapReduce technology in a cluster [38].

Hadoop and its machine learning library Mahout have been selected for classification tasks using Random Forest in [39]. Meta classifiers combined automatically in an iterative way have been proposed for the detection of malware in [40]. A classifier ensemble algorithm for multimedia classification was proposed in [41], where a decision tree was used to combine the predictions of the individual ensemble members. However, an extensive analysis of the literature reveals that these methods have not been applied to the prediction of big data time series, and therefore, the work here introduced attempts at filling this gap.

Recently, to alleviate some of the issues associated with big data, Do and Poulet [42] developed a parallel ensemble learning algorithm of random local SVM that was able to perform much better than the standard SVM algorithm.

As a large number of resources is necessary to generate the ensemble, in [43] a new ensemble method that minimizes the usage of resources was proposed. Learning a decision multi-tree instead of a decision tree allows to share the common parts of the components of the ensemble and build a shared ensemble.

After a thorough review of the previously published works, it can be concluded that forecasting of big data time series is an emerging topic that should be further investigated. In particular, very few papers have been published using distributed and highly scalable computing systems. Additionally, not many ensemble methods for big data have been proposed and none of them made use of the Spark benefits. In this paper we aim to address these limitations by proposing and investigating the performance of ensemble method for big data forecasting, that makes use of the Spark engine.

In the next section we describe our methodology for forecasting big data time series using static and dynamic ensemble models.

3. Methodology

In this section, we firstly introduce the three regression methods that we use to generate prediction models for big data time series. Then, we present the proposed ensemble model which combines the individual tree-based regression models to further improve the prediction accuracy. For all regression methods we use their MLlib library implementation, to ensure the scalability of the ensemble model, and therefore, its ability to deal with big data time series.

3.1. Decision tree

Decision Trees (DTs) are a very popular and successful machine learning method, for both classification and regression tasks. Some of the advantages they offer are the following: they are able to model nonlinear relations between the attributes and the target variable, do not require attribute scaling, and the resulting tree (a set of if-then rules) is easy to interpret and use for decision making by the end-user.

A DT is built through a recursive binary partition of the feature space. A node in the tree corresponds to a test for the value of an attribute and a leaf node corresponds to a regression function. When building the tree, at each step the attribute with the highest information gain is selected. A test for its value is used to split the tree, creating a branch for the possible outcomes. The test divides the instances into several subsets, based on the attribute value. The process is repeated recursively for each subset until the stopping condition is satisfied, in which case a leaf node is created. The tree growing stops when there is no split candidate with sufficiently high information gain or when a pre-specified maximum tree depth is reached.

To predict the value for a new instance, we start at the root of the tree and follow the path corresponding to the values of the instance until a leaf node is reached and the prediction is obtained.

MLlib supports DTs for both classification and regression, and can be used with both continuous and discrete attributes.

3.2. Gradient boosted trees

Gradient Boosted Trees (GBT) is an ensemble of DTs. An ensemble method combines the predictions of a set of base models. DT-based ensembles such as GBT have showed high performance in regression and classification tasks [44,45]. GBT creates the tree ensemble iteratively. Thus, the errors made by the first tree are taken into account when adding the second tree and so on. The final prediction is the mean of the predictions of the individual trees.

3.3. Random forests

Random Forest (RF) is also an ensemble of DTs. In contrast to GBT, where the trees are built iteratively, RF trains multiple trees in parallel. Each tree uses a different training set generated by creating a bootstrap sample from the training data. In addition, when selecting the best attribute at each node, only a subset of all attributes available at the node will be considered, instead of all features as in DTs. Thus, RF uses both random instance and feature selection. To make a prediction for a new instance, RF takes the average of the predictions obtained from each tree.

In summary, both GBT and RF use decision trees as a base model, but the training process is different and each of the two methods has its advantages and disadvantages. MLlib supports both GBT and RF.

3.4. Proposed ensemble model

Ensembles of prediction models are one of the most successful methods used in practical applications [34]. An ensemble usually improves the results of the single base models it combines. In this paper we focus on ensembles for time series forecasting, and especially for big data. We propose to combine DT, GBT and RF in an ensemble, due to the good results obtained by each of these algorithms when applied to big data time series forecasting [15].

Instead of combining the predictions by taking the average of the individual predictions, which is the most simple and straightforward combination, we propose to use a weighted average combination, where different weights for each algorithm are computed based on its previous performance.

To calculate the coefficients for the weighted prediction, we minimize the forecasting error on a validation set. Let K be the number of algorithms that form the ensemble model. Let us suppose that the validation set is composed of N instances and h is the prediction horizon. Then, a weighted least squares method is applied to minimize the squared error between the predictions of the K algorithms and the actual values for the N instances of the validation set. Thus, the weights that minimize the difference between the predicted and actual values are obtained by solving the following equation for each jth value of the prediction horizon:

$$\widehat{P}^j \alpha^j = b^j \tag{1}$$

where $\widehat{P}^j$ is a matrix with N rows and K columns containing the prediction for the jth value of the prediction horizon for the validation set for each algorithm, α^j is a vector of K elements corresponding to the weights for the jth value of the prediction horizon for each algorithm, and b^j is a vector composed of the N actual values for the jth value of the prediction horizon for the validation set. The predictions $\widehat{P}^j$ for each algorithm are computed following the methodology described in the Section 3.5.

Once the weights have been obtained by the weighted square least method, we use them to make predictions for the test set. Let us suppose the test set is composed of M instances. Then, the Mxh predictions represented by the matrix $\widehat{P}$ are computed by a linear combination of the predictions for the K algorithms, where the coefficients of the linear function are given by the weights:

$$\widehat{P} = \left[\widehat{Q}^1 \alpha^1, \ldots, \widehat{Q}^h \alpha^h\right] \tag{2}$$

where $\widehat{Q}^j$ is a matrix with M rows and K columns containing the prediction for the jth value of the prediction horizon for the test set for each algorithm and α^j are the weights obtained by Eq. (1). Thus, $\widehat{Q}^j \alpha^j$ is a vector of M elements containing the predictions obtained by the ensemble model for the jth value of the prediction horizon for the test set.

From Eq. (2) it also follows that:

$$\widehat{p}_{i,j} = \sum_{l=1}^{K} \alpha_l^j \widehat{p}_{i,l}^j \tag{3}$$

where $\widehat{p}_{i,j}$ is the (i,j)-th element of the matrix $\widehat{P}$, $\widehat{p}_{i,l}^j$ is the (i,l)-th element of the matrix $\widehat{Q}^j$ and α_l^j is the lth element of the vector α^j.

Both Eqs. (2) and (3) represent a static ensemble combination, i.e. the same weights are used to predict all Mxh values of the test set. However, different adaptive strategies can be used to update the weights after a given time interval, and thus to create dynamic ensembles.

Fig. 1 shows a general diagram of the static ensemble model described above to predict a big data time series. Note that the weights are defined by a matrix due to a multi-step prediction problem that we address in this paper.

In the dynamic ensemble model, let R be the updating period for the weights. Then the test set TS can be divided into subsets as follows:

$$TS = \bigcup_{t=1}^{R} TS^t \tag{4}$$

where the subset TS^t is composed of [M/R] instances and $[\cdot]$ denotes the whole number part of the division. Then, the weights are found by solving the following equation:

$$\widehat{P}^j(t)\alpha^j(t) = b^j(t) \quad t = 1, \ldots, R \tag{5}$$

where $P^j(t)$ and $b^j(t)$ are the predictions and actual values for the jth value of the prediction horizon respectively, using the validation set from the training set updated as follows:

$$TRS \leftarrow TRS^t \bigcup \left(\bigcup_{l=1}^{t-1} TS^l \right) \quad t = 1, \ldots, R \tag{6}$$

where TRS^t is the training set by removing the oldest $(t-1)[M/R]$ instances. Note that $(t-1)[M/R]$ is the size of the $\bigcup_{l=1}^{t-1} TS^l$, and therefore, the updating of the training set consists of sliding the training set $(t-1)[M/R]$ instances forward while keeping the same training set size.

Once the weights have been obtained by the weighted square least method from Eq. (5), the predictions for each subset TS^t are computed as:

$$\widehat{P}(t) = \left[\widehat{Q}^1(t)\alpha^1(t), \ldots, \widehat{Q}^h(t)\alpha^h(t) \right] \quad t = 1, \ldots, R \tag{7}$$

where $\widehat{Q}^j(t)$ contains the prediction for the jth value of the prediction horizon for the TS^t test subset for each algorithm and $\alpha^j(t)$ are the weights obtained by Eq. (5).

Fig. 2 presents graphically how the dynamic ensemble model works when the weights are periodically updated, in our case, every 3 months.

3.5. Multi-step forecasting

This section summarizes the forecasting methodology proposed in our previous work [15] for h-steps ahead prediction for big data time series using the MLlib library of Apache Spark.

Problem formulation. Given a time series with previous values up to time t, $[x_1, \ldots, x_t]$, the task is to predict the h next values of the time series, from a window of w past values, as shown in Fig. 3.

This forecasting problem can be formulated as below, where f is the model to be learnt by the forecasting method in the training phase:

$$[x_{t+1}, x_{t+2}, \ldots, x_{t+h}] = f\left(x_t, x_{t-1}, \ldots, x_{t-(w-1)}\right) \tag{8}$$

However, the existing regression techniques in MLlib do not support this multi-step forecasting. Therefore, we split the problem into h forecasting sub-problems as follows:

$$\begin{aligned} x_{t+1} &= f_1\left(x_t, x_{t-1}, \ldots, x_{t-(w-1)}\right) \\ x_{t+2} &= f_2\left(x_t, x_{t-1}, \ldots, x_{t-(w-1)}\right) \\ &\vdots \quad = \quad \vdots \\ x_{t+h} &= f_h\left(x_t, x_{t-1}, \ldots, x_{t-(w-1)}\right) \end{aligned} \tag{9}$$

Hence, in each sub-problem we use the same input data with size w but predict a different target value from the forecasting window h. We note that using this formulation, the existing possible relations between the h consecutive values $x_{t+1}, \ldots, x_{t+h}$ are not taken into consideration. An alternative approach would be to use a rolling forecasting where the predictions of the previous values are used as actual values when predicting the next value, e.g. after predicting x_{t+1}, it will be included in the w values used to predict x_{t+2}. However, we found that the rolling forecasting method was less accurate due to the accumulation of the error along the prediction horizon. Hence, we chose the first approach.

We build h different training sets. Each training instance is composed of the w features. The target value for each of the h problems corresponds to a different value of the prediction horizon, as shown in Fig. 3. Thus, we learn h prediction models.

To predict a new instance from the test data, the prediction of the ith value of the prediction horizon is obtained by using the ith model with the corresponding w features from the test set as an input.

This methodology was tested in [15] using four different regression methods from MLlib (linear regression, DT, GBT and RF) demonstrating the suitability of these methods for big data time series forecasting.

In the next section we evaluate the performance of the proposed ensemble models on Spanish electricity consumption data for 10 years measured with a 10-minute frequency.

4. Results for electricity consumption data

In this section, we present and discuss the application of our proposed method for prediction of big electricity consumption time series data. We firstly describe the electricity consumption dataset in Section 4.1. The experimental setting is presented in Section 4.2 and the sensitivity analysis used to select an adequate historical window size is provided in Section 4.3. We present and analyse the results obtained by the different static and dynamic ensemble methods in Section 4.4.

4.1. Dataset description

The time series used in this work is related to the total electrical energy consumption in Spain, from January 1st 2007 at midnight to June 21st 2016 at 11:40 pm. In short, it is a time series of nine and a half years with a high sampling frequency, namely 10 min intervals, including 49,832 measurements in total.

When using the proposed methodology with a prediction horizon of 4 h (h is set to 24 values), the dataset consists of 20,742 instances and 144 attributes, corresponding to 5.70 MiB of storage size. These 144 attributes correspond to a window w of 144 past values (24 h).

For the static ensemble, this dataset is divided into a training set and a test set consisting of 60% and 40% of the data, respectively. The training set has 298,752 measurements; it includes data from January 1st, 2007 at midnight to September 8th, 2012 at 10:30 am. The test set contains the remaining data, namely 199,080 measurements from September 8th, 2012 at 10:40 am to June 21st, 2016 at 11:40 pm.

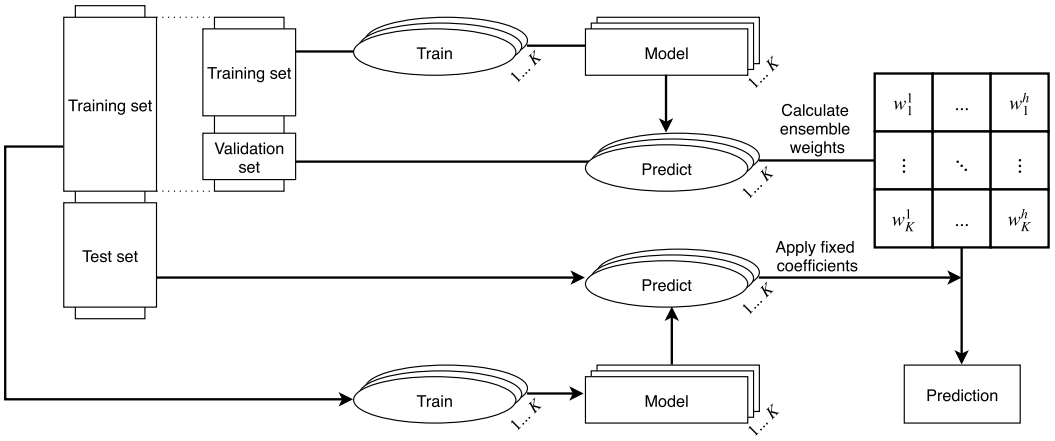


Fig. 1. Workflow of the proposed static ensemble.

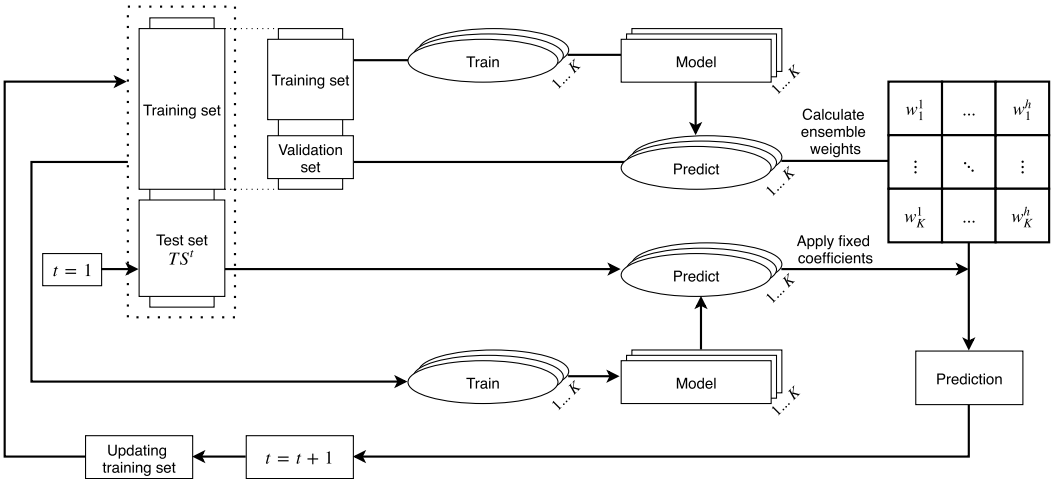


Fig. 2. Workflow of the proposed dynamic ensemble.

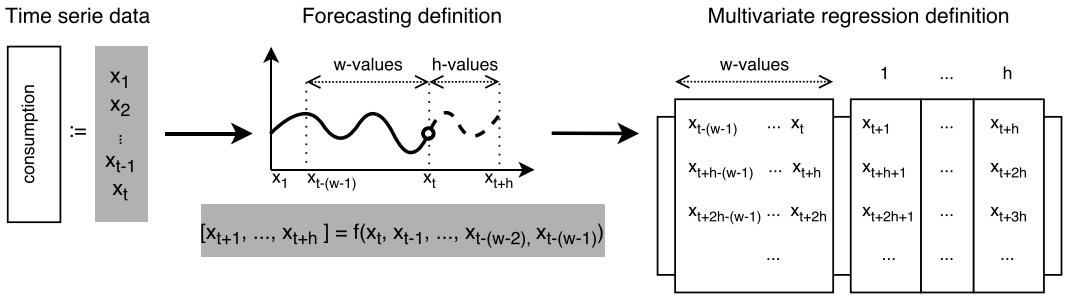


Fig. 3. Illustration of the multivariate problem.

The training set is divided again into a sub-training set –60% used to generate the prediction model for each algorithm , – and a validation set — the remaining 40% used to obtain the weights of the ensemble method .

In the case of dynamic ensemble, the weights of the ensemble are updated every 3 months (every 13,104 predicted values) sliding the training set 13,104 measurements forward while keeping the same training set size. In the same way, the prediction model is updated every 3 months.

4.2. Design of experiments

The experimentation carried out consists of a total of 248 executions, obtaining a total of 5952 prediction models for the time series of electrical consumption in the Spanish electricity market. The experimental setting is summarized below:

1. The size of the window w formed by past values has been set to 24, 48, 72, 96, 120, 144 and 168, corresponding to 4, 8, 12, 16, 20, 24 and 28 h, respectively. Given this number of past values, the goal is to predict the next 24 values.
2. The number of trees and the maximum depth of trees are input parameters in GBT and RF. Both parameters were tested in [15] and the optimal configuration obtained is used in this work. Specifically, a depth of 8 has been used for both algorithms, 5 trees for GBT and 100 trees for RF.
3. The ensemble technique combines DT, GBT and RF. When a dynamic ensemble is applied, the weights are updated every

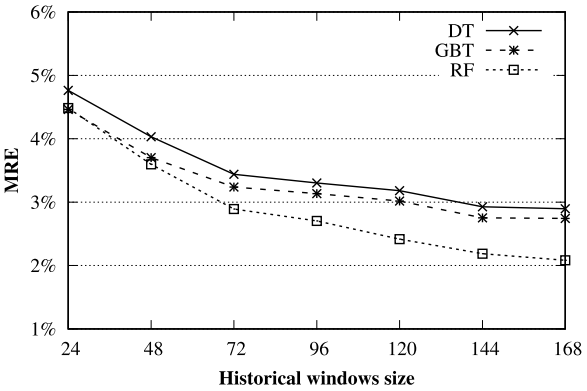


Fig. 4. MRE evolution for different historical window sizes.

3 months. Thus, the dynamic ensemble uses a total of 2304 prediction models.

The mean relative error (MRE) has been used as an evaluation measure to compare the accuracy of the predictions obtained by the different prediction methods. The MRE is defined as follows:

$$MRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|p_i - a_i|}{a_i} \tag{10}$$

where a_i and p_i represent actual and predicted values of the time series, respectively, and n is the number of samples to be predicted.

The experimentation was conducted using high-performance computing resources on the Open Telekom Cloud Platform with five machines, one master and four slave nodes. Each node has 60 GB of main memory and 8 logical cores from an Intel Xeon E5-2658 v3 @ 2.20 GHz processor that has 30 MB L3 cache. The cluster works with Apache Spark 2.1.2 and Hadoop 2.6.

4.3. Sensitivity analysis

This section presents a sensitivity analysis regarding the size of the historical window for the DT, GBT and RF algorithms, which are included in the ensemble model. The analysis consists of a total of 152 executions, obtaining 3648 prediction models in total.

Fig. 4 shows the evolution of MRE on the validation set when increasing the window size for the three proposed methods. For all methods, we can see an improvement in MRE as the size of the window w increases. For the DT algorithm, the optimal configuration was obtained with a window of 168 values, resulting in MRE of 2.90%. For GBT, the optimal model used a window of 168 past values obtaining MRE of 2.74%. Finally, for RF, the smallest error of 2.08% was obtained with a window of 168 past values. However, we can see that increasing the window size from 144 to 168 does not lead to a significant improvement for DT and GBT.

Based on the analysis above, $w = 144$ has been selected for the results shown in the following sections. We note that this window size is not accidental – it represents the values corresponding to the past 24 h –demonstrating the strong stationarity of the time series of electric demand during the day.

4.4. Analysis of results

In this section, we present and discuss the accuracy of the ensemble prediction models – the daily errors along with the worst and best days and the average relative errors of the static and dynamic ensemble models on the test set –comparing them to the errors of the single DT, GBT and RF models.

Table 1
MRE (%) distribution.

Interval	Static	(Agg)	Dynamic	(Agg)
[0,0.5)	0.00	(0)	0.00	(0)
[0.5,1)	0.87	(1)	5.50	(5)
[1,1.5)	13.82	(15)	30.82	(36)
[1.5,2)	28.58	(43)	22.79	(59)
[2,2.5)	23.52	(67)	17.80	(77)
[2.5,3)	16.86	(84)	9.41	(86)
[3,3.5)	6.73	(90)	5.72	(92)
[3.5,4)	3.62	(94)	2.46	(95)
[4,4.5)	2.60	(97)	2.75	(97)
[4.5,5)	1.45	(98)	0.51	(98)
[5,9.5)	1.95	(100)	2.24	(100)

Recall that in the case of DT, GBT, RF and the static ensemble model, we build one prediction model by using 60% of the data as training set. For the dynamic ensemble model, the training set always retains the same size, but the model is updated every 3 months, i.e., every 13,104 values. Thus, the training set is slid forward 13,104 measurements and the weights of the ensemble model are calculated again from the new validation set. Then, a new updated model is obtained to predict the 13,104 next values.

4.4.1. Overall performance

Fig. 5 presents the mean relative error for the static and dynamic ensemble and each individual model they combine, for every hour of the 24 h forecasting horizon. Table 3 also shows the aggregated mean values for each prediction algorithm for the whole test set. From Table 3 we can see that the most accurate prediction model is the dynamic ensemble –it outperforms the static ensemble and all other prediction models . This shows the benefits of dynamically adapting to the changes in the time series when building prediction models.

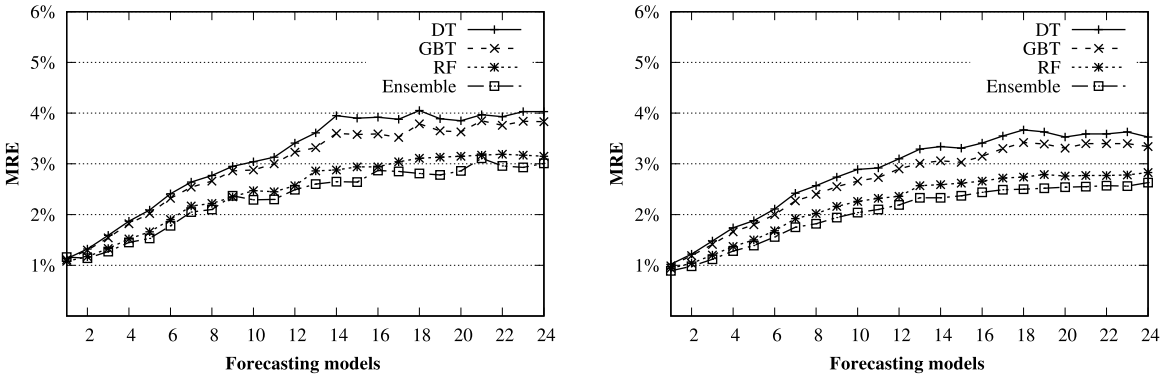
Within each group (dynamic and static), the ensemble outperforms the individual prediction models it combines. The most accurate individual prediction model is RF, followed by GBT and DT. DTs are single classifiers, so it is expected that they will be outperformed by ensembles of trees such as GBT and RF. RF is performing very well showing the advantage of using two strategies for generating diverse ensemble members — bagging and random feature selection when selecting the best attribute .

Fig. 5 shows that initially (during the first 1–2 h of the forecasting horizon) all methods perform similarly. For hours 3–7, RF and the ensemble show similar performance and start outperforming the other methods, and after that the ensemble method outperforms RF. For the following hours of the forecasting horizon, the ranking of the algorithms is consistent (ensemble, RF, GBT and DT).

From Fig. 5 we can also see that MRE increases as the forecasting horizon increases which is as expected. Thus, the lowest and highest MRE are obtained when the first and last value of the prediction horizon are forecasted, respectively.

4.4.2. Daily performance

To study the daily MRE we group the predictions of each algorithm into groups of 144 values as the measurements are taken every 10 min and we predict 24 h ahead. Fig. 6 shows the histogram of the daily MRE of the test set for the dynamic and static ensemble. The histogram represents the frequency of the daily MRE in different intervals when predicting all days in the test set. We can see that the dynamic ensemble considerably reduces the error in the intervals between 0.5% and 1.0% and 1.0% and 1.5%, where the accuracy is the highest. The impact of this improvement is also noticeable for daily MRE between 1.5% and 3%, due to the low number of days with these average prediction errors for the dynamic ensemble model.



(a) Static ensemble (b) Dynamic ensemble

Fig. 5. MRE for each model, for each time point of the prediction horizon.

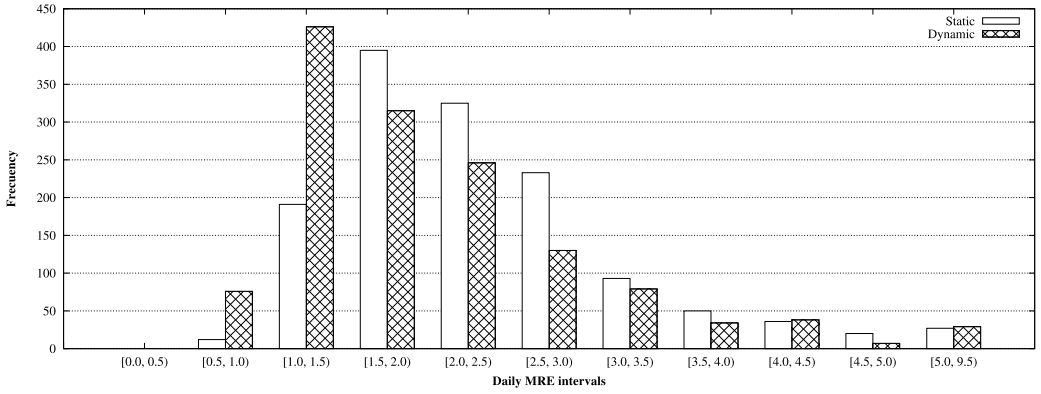


Fig. 6. Histogram of daily errors for static and dynamic ensemble models.

The percentage along with the accumulated number of days for each interval in the histogram is shown in Table 1. As it can be seen, the proposed dynamic ensemble model is stable as the 98% of days have a MRE less than 5% and all days exceeding this threshold correspond to holidays or long weekends as Table 2 shows.

4.4.3. Worst and best days

It is also interesting to study the worst and best predicted days for the different forecasting methods. Table 3 presents the MRE for the best and worst predicted day for DT, GBT, RF and also the static and dynamic ensembles. As it can be observed, the static ensemble model improves the MRE about 25% compared to DT, 21% compared to GBT and 6% compared to RF. In the case of the dynamic ensemble, it achieves a MRE improvement of 28% compared to DT, 23% comparing to GBT and 8% compared to RF. The dynamic ensemble is clearly the best performing model, outperforming the static ensemble with 13%. In addition, it can be noticed that the three ensemble models have similar prediction error variance, which is lower than the variance of the individual methods.

Table 4 compares the static and dynamic ensembles with an ANN that supports the multi-output regression. An ANN configuration with 84 neurons in the hidden layer (the mean of input and output neurons) and a hyperbolic tangent activation function has been selected. It can be seen that the accuracy of ANN is lower than both ensemble models. Compared to the dynamic ensemble, the MRE of ANN for the best predicted day – 2.0199% – is much bigger than the error of the ensemble — 0.7189% . Similarly, the error of the worst predicted day increases from 8.6016% for the ensemble to 17.0503% for ANN.

Fig. 7 shows a graphical representation of the MRE for the test set, and also the MRE corresponding to the days with the best and

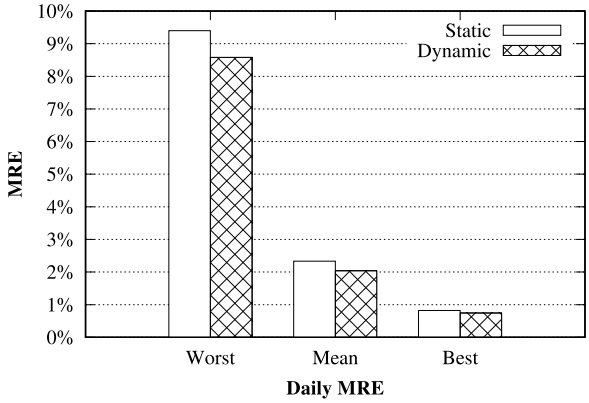


Fig. 7. Comparison of daily MRE of static and dynamic ensemble models.

the worst prediction, for each ensemble model. We can see again that the dynamic ensemble outperforms the static ensemble in all three cases.

Fig. 8 shows how the weight of each individual model in the ensemble is distributed over time. It can be seen that the weights of all three models remain stable for the prediction horizon considered. Moreover, the contribution of the single models ranges from 20% to 40% on average, showing that there is no single dominating model.

Fig. 9 provides information about the hourly predictive performance of the static and dynamic ensemble for the best day. Specifically, Fig. 9(a) shows the actual and predicted electricity demand for the day with the best prediction (MRE of 0.82%), obtained by

Table 2
Worst days for static and dynamic ensemble models.

Static ensemble			Dynamic ensemble		
MRE	Day	Type of Day	MRE	Day	Type of Day
9.32	24/12/13	Christmas Eve	8.60	24/12/13	Christmas Eve
7.71	24/12/12	Christmas Eve	7.18	19/04/14	Easter
7.66	19/04/14	Easter	7.16	30/03/13	Easter
7.40	30/03/13	Easter	7.01	29/03/13	Easter
7.31	24/12/15	Christmas Eve	6.98	24/12/12	Christmas Eve
7.15	31/12/12	New Year's Eve	6.88	31/12/12	New Year's Eve
7.14	24/12/14	Christmas Eve	6.71	24/12/15	Christmas Eve
6.96	31/12/15	New Year's Eve	6.33	31/03/13	Easter
6.56	31/03/13	Easter	6.00	30/04/14	Labour Day
6.41	30/04/13	Labour Day	5.83	30/04/15	Labour Day
6.29	17/04/14	Easter	5.72	01/04/13	Easter
6.29	31/12/13	New Year's Eve	5.58	31/12/15	New Year's Eve
6.00	30/04/15	Labour Day	5.54	31/12/13	New Year's Eve
5.97	29/03/13	Easter	5.54	07/12/14	Immaculate Conception
5.95	21/04/14	Easter	5.48	26/12/12	Christmas Day

Table 3
MRE (mean and variance) on the test set, for the worst and best predicted days.

Model	Algorithm	Worst (%)	Mean (%)	Variance (%)	Best (%)
Static	DT	10.2102	3.1400	0.014	1.2401
	GBT	10.1950	2.9680	0.012	1.2074
	RF	8.8475	2.4838	0.011	0.7621
	Ensemble	**9.3207**	**2.3320**	**0.009**	**0.8230**
Dynamic	DT	9.5022	2.8395	0.015	1.1500
	GBT	9.1633	2.6569	0.014	1.0782
	RF	8.5162	2.2243	0.012	0.6530
	Ensemble	**8.6016**	**2.0362**	**0.010**	**0.7189**

Table 4
Static and dynamic ensembles comparison with ANN.

Method	Worst (%)	Mean (%)	Variance (%)	Best (%)
ANN	17.0503	4.0342	0.136	2.0199
Static ensemble	9.3207	2.3320	**0.009**	0.8230
Dynamic ensemble	**8.6016**	**2.0362**	0.010	**0.7189**

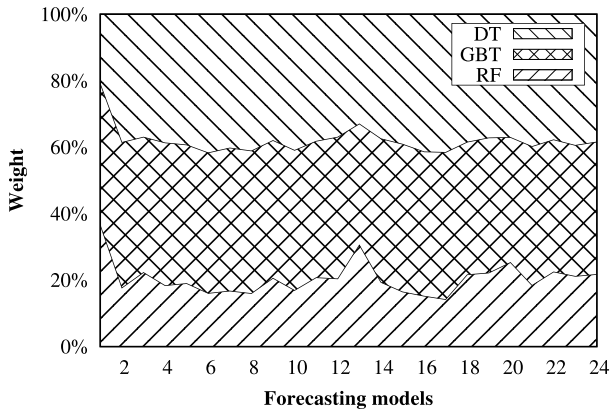


Fig. 8. Static ensemble — weight change during the prediction horizon for the individual models .

the static ensemble model. This day corresponds to the 24 h from Tuesday, August 4th, 2015 at 10:00 pm until Wednesday, August 5th, 2015 at 10:50 pm. Fig. 9(b) shows the same information for the day with the best prediction (MRE of 0.74%), obtained by the dynamic ensemble model. This day, corresponds to the 24 h from Wednesday July 30th, 2014 at 11:00 pm until Thursday July 31st, 2014 at 10:50 pm.

It is also important to analyse the days with worst predictions since they contribute to increasing the average errors. Fig. 10(a) shows the day with the worst prediction obtained with the static ensemble model, resulting in a MRE of 9.32%. This day corresponds to the 24 h from Tuesday December 24th, 2013 at 11:00 pm to Wednesday December 25th at 10:50 pm. Fig. 10(b) shows the day with the worst prediction obtained with the dynamic ensemble model, resulting in MRE of 8.60%, also for the 24 h from Tuesday December 24th, 2013 to Wednesday December 25th. This coincidence is reasonable because December 24th and 25th are special days (Christmas holidays) characterized by more random electricity consumption; they are more different than the previous days, and hence, are more difficult to predict.

In summary, our results showed that both the static and dynamic ensembles performed well and were more accurate than the individual prediction models they combined. The dynamic ensemble was considerably more accurate than the static ensemble, for all predicted days from the test set, achieving an improvement in MRE of about 13%. It also reduced the error of the worst predicted day by 8% and the error of the best predicted day by 13%.

5. Results for solar photovoltaic data

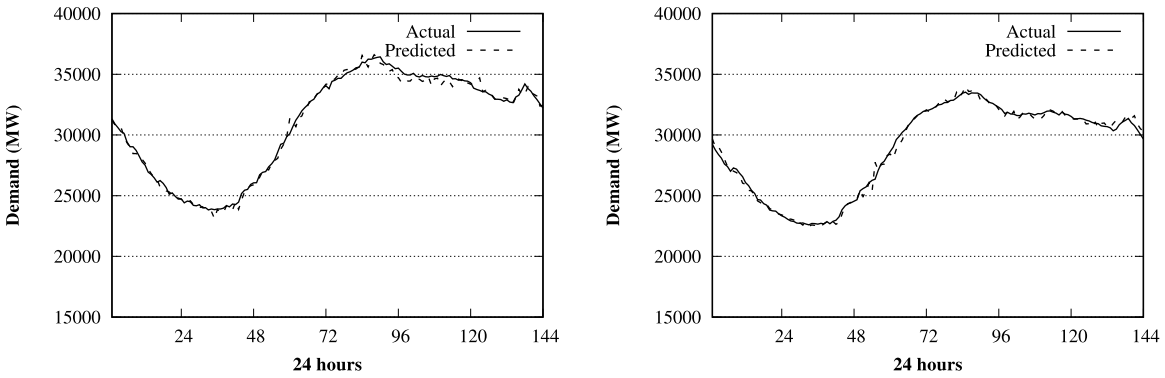
Solar energy is a very promising renewable energy source, which is still underutilized. However, in recent years there has been a considerable increase in production worldwide. Solar energy production depends on weather conditions such as solar radiation, cloud cover, rainfall and temperature. This dependence creates uncertainty where it is important to ensure a reliable supply of electricity, making it difficult to integrate solar energy into electricity markets. Therefore, the ability to predict the solar energy generated is a critical task for stakeholders in the energy sector.

In this section, we present and discuss the application of our proposed method for prediction of solar power data. We firstly describe the dataset in Section 5.1. The experimental setting is presented in Section 5.2. We present and analyse the results obtained by the different static and dynamic ensemble methods in Section 5.3.

5.1. Dataset description

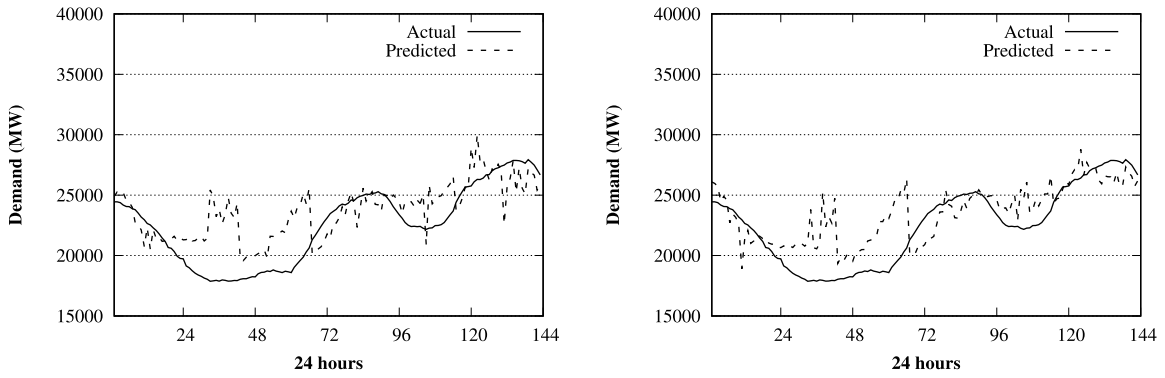
The time series used is related to Australian solar photovoltaic data for two years, from January 1st 2015 to 31 Decemberst 2016. It is a time series with 30 min intervals, where each day has 20 measurements corresponding to the solar day from 7:00 to 17:00.

When using the proposed methodology with a prediction horizon of 10 h (h is set to 20 values), the dataset consists of 730 instances and 20 attributes. These 20 attributes correspond to a window w of 20 past values (10 h). This dataset is divided into a training set and a test set consisting of 70% and 30% of the data, respectively.



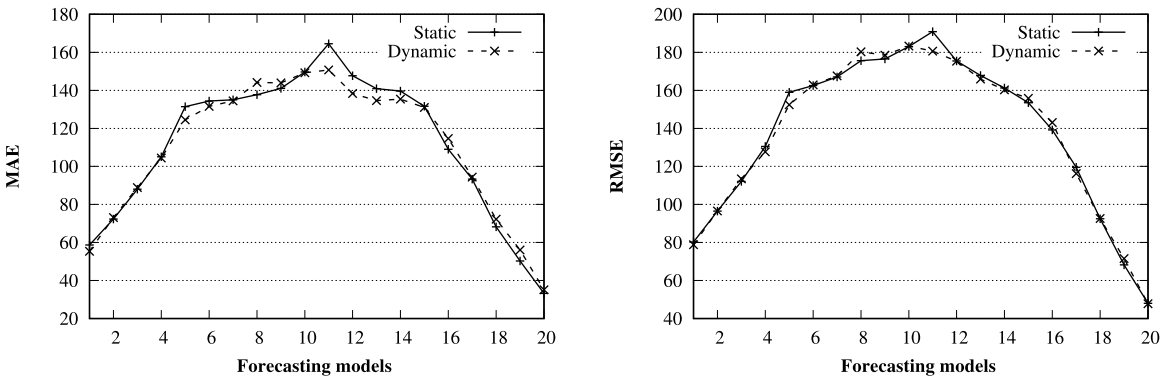
(a) Static ensemble (b) Dynamic ensemble

Fig. 9. Best predicted day — actual and predicted values .



(a) Static ensemble (b) Dynamic ensemble

Fig. 10. Worst predicted day — actual and predicted values .



(a) MAE comparison (b) RMSE comparison

Fig. 11. MAE and MRSE comparison for each time point of the prediction horizon.

The training set is divided again into a sub-training set – 30% used to generate the prediction model for each algorithm –and a validation set—the remaining 30% used to obtain the weights of the ensemble method.

In the case of dynamic ensemble, the weights of the ensemble are updated every two weeks (every 280 predicted values) sliding the training set 280 measurements forward while keeping the same training set size. In the same way, the prediction model is updated every two weeks.

5.2. Design of experiments

Deep learning (DL) has been used to forecast large datasets of solar energy data [46], comparing the performance with two other advanced forecasting methods published in [47]. In particular, Pattern Sequence-based Forecasting (PSF) based on pattern similarity [48] and an Artificial Neural Network (ANN). Static and dynamic ensembles are compared with these algorithms.

According to the referenced work to compare with, the experimental setting is summarized below:

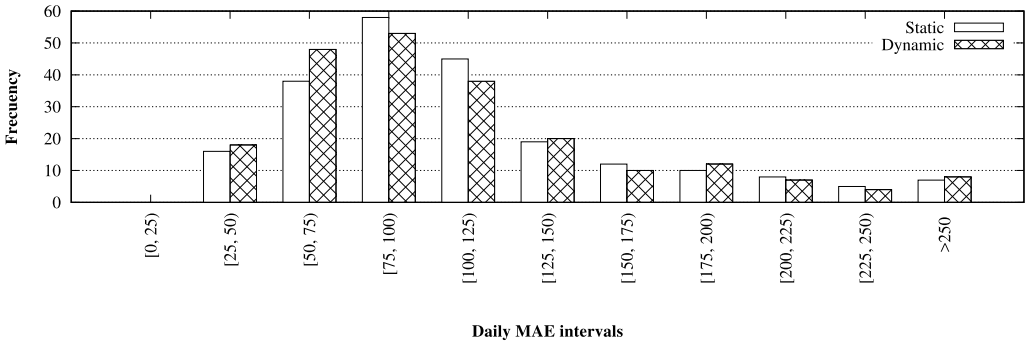


Fig. 12. Histogram of daily errors for static and dynamic ensemble models.

1. The size of the window w is formed by 20 past values, corresponding to 10 h. Given this number of past values, the goal is to predict the next 20 values.
2. The number of trees and the maximum depth of trees are input parameters in GBT and RF. Specifically, a depth of 8 has been used for both algorithms, 5 trees for GBT and 100 trees for RF.
3. The ensemble technique combines DT, GBT and RF. When a dynamic ensemble is applied, the weights are updated every two weeks.

The mean absolute error (MAE) and root mean squared error (RMSE) have been used as evaluation measures to compare the accuracy of the predictions obtained by the different prediction methods. MAE and RMSE are defined as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i| \tag{11}$$

$$RMSE = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(p_i - a_i)^2} \tag{12}$$

where a_i and p_i represent actual and predicted values of the time series, respectively, and n is the number of samples to be predicted.

5.3. Analysis of results

In this section, we present and discuss the accuracy of the ensemble prediction models – the daily errors along with the worst and best days and the average relative errors of the static and dynamic ensemble models on the test set –comparing them to the errors of the ANN, PSF and DL algorithms.

Recall that in the case of the static ensemble model, we build one prediction model by using 70% of the data as training set. For the dynamic ensemble model, the training set always retains the same size, but the model is updated every two weeks, i.e., every 280 values. Thus, the training set is slid forward 280 measurements and the weights of the ensemble model are calculated again from the new validation set. Then, a new updated model is obtained to predict the 280 next values.

5.3.1. Overall performance

Fig. 11 presents the MAE and RMSE for the static and dynamic ensemble, for every half hour of the 10-hour forecasting horizon. From Table 5 we can see that the most accurate prediction model is the dynamic ensemble. It outperforms the static ensemble and all other prediction models.

Fig. 11 shows that initially (during the first 1–3 h of the forecasting horizon) the accuracy is high. For hours 4–7, prediction is more difficult, because error increases. For the following hours of the forecasting horizon, both methods perform better again.

Table 5
Comparison of the performance of solar energy forecasts.

	Worst		Mean		Best	
Method	MAE	RMSE	MAE	RMSE	MAE	RMSE
ANN	**191.52**	**221.58**	114.64	154.16	58.87	106.88
PSF	252.77	279.12	117.17	147.52	31.72	36.15
DL	206.33	233	114.76	148.98	31.66	41.91
Static ensemble	303.83	353.45	111.59	130.98	**25.93**	**32.95**
Dynamic ensemble	329.60	362.60	**110.62**	**129.46**	26.72	34.04

These result are consistent for both MAE (Fig. 11(a)) and RMSE (Fig. 11(b)), where static and dynamic ensemble behaviours are similar.

5.3.2. Daily performance

To study the daily MAE and MRSE we group the predictions of each algorithm into groups of 20 values as the measurements are taken every 30 min and we predict next day ahead. Fig. 12 shows the histogram of the daily MAE and MRSE of the test set for the dynamic and static ensemble. The histogram represents the frequency of the daily MAE and MRSE in different intervals when predicting all days of the test set. We can see that the dynamic ensemble considerably reduces the error in the MAE intervals between 75 and 125, where the accuracy is the highest.

5.3.3. Worst and best days

It is also interesting to study the worst and best predicted days for the different forecasting methods. Table 5 presents the MAE and MRSE for the best and worst predicted day for ANN, PSF, DL and also the static and dynamic ensembles. As it can observed, the dynamic ensemble achieves the best accuracy by average, and the static ensemble the predicted best day.

Fig. 13 shows a graphical representation of the MAE and RMSE for the test set, and also the MRE corresponding to the days with the best and the worst prediction, for each ensemble model. We can see similar behaviours between dynamic and static ensembles.

It is also important to analyse the days with worst predictions since they contribute to increase the average errors. Fig. 14(a) shows the day with the worst prediction obtained with the static ensemble model, resulting in a MAE of 303.83. Fig. 14(b) shows the day with the worst prediction obtained with the dynamic ensemble model, resulting in MAE of 329.60.

Fig. 15 provides information about the predictive performance of the static and dynamic ensemble for the best day. Specifically, Fig. 15(a) shows the actual and predicted PV data for the day with the best prediction (MAE of 25.93), obtained by the static ensemble model. Fig. 15(b) shows the same information for the day with the best prediction (MAE of 26.72), obtained by the dynamic ensemble model.

The results show that these ensemble methods offer accurate predictions, obtaining better averaged results that the methods

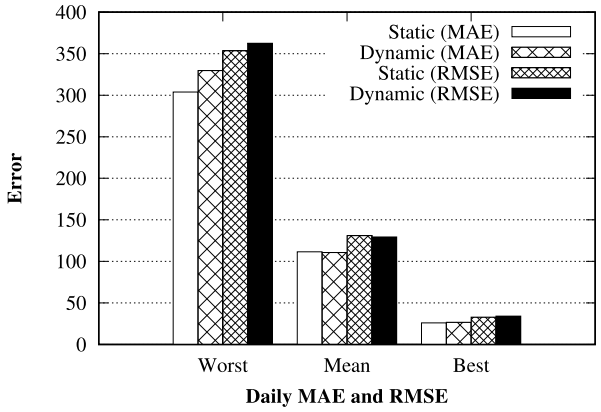


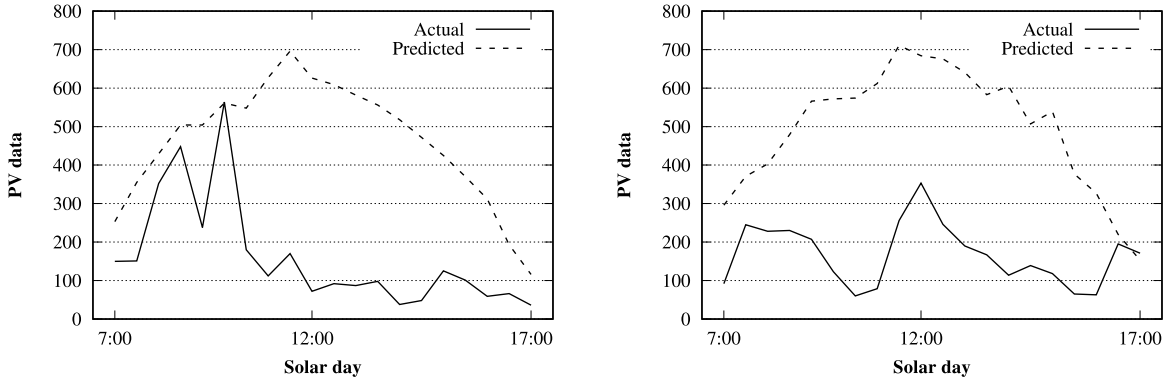
Fig. 13. Comparison of daily MAE and RMSE of static and dynamic ensemble models.

used for comparison. When the day to predict is very atypical, ensembles also have difficulty for obtaining accurate predictions. However, the best predicted days by the ensembles have a higher accuracy than ANN, PSF and DL algorithms. A bigger dataset with more training instances could help the dynamic ensemble.

6. Conclusions

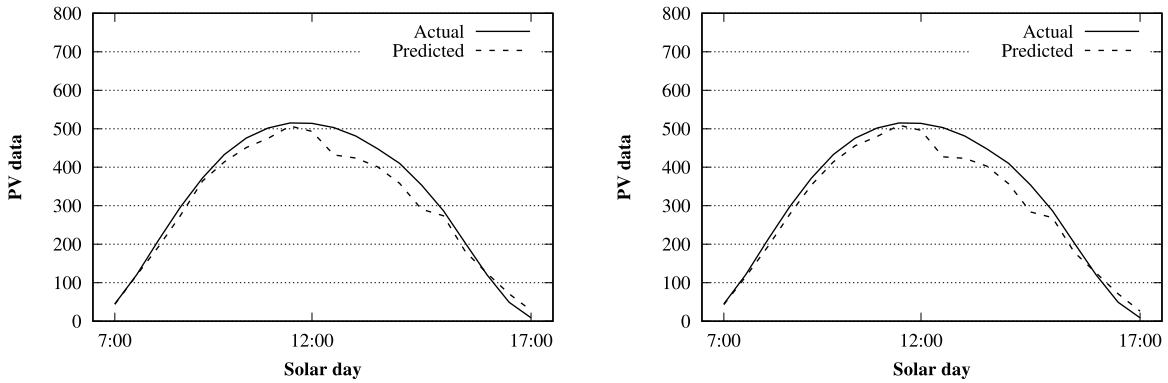
In this paper, we proposed a novel approach based on ensemble learning for predicting big data time series (time series with a high sampling frequency and multi-step forecast horizon). We proposed an ensemble method which computes the weights for each ensemble member using a least square method, assigning higher weights to the more accurate ensemble members based on their past performance. We investigated two strategies for updating the weights, resulting in a dynamic and static ensemble. Although in our case study we have chosen to combine tree-based regression models (DT, GBT and RF), our method can be used to combine other type of prediction models. For the implementation of the prediction algorithms we have used the MLlib library of the Apache Spark framework, to ensure the scalability of our method and its suitability for big data. We conducted a comprehensive evaluation using Spanish electricity consumption data for 10 years consisting of about 500,000 measurements at 10-min intervals. Our results showed that both ensemble methods performed well, outperforming the individual ensemble members they combined, but the dynamic ensemble was the best method. It considerably outperformed the static ensemble, obtaining MRE of 2%. This is very competitive result, showing the viability of the proposed methodology for the prediction of big time series. In addition, Australian solar data have been used to compare static and dynamic ensembles with ANN, PSF and DL, improving the accuracy of these algorithms.

In future work, we plan study the addition of other types of prediction models to the ensemble, which are suitable for big data, in order to increase the diversity among the ensemble members. We will also investigate other machine learning strategies for determining the weights in a dynamic way. We also plan to evaluate the performance of our dynamic ensemble on other big datasets from different sources. Finally, we will develop models for forecasting special days.



(a) Static ensemble (b) Dynamic ensemble

Fig. 14. Worst predicted day—actual and predicted values .



(a) Static ensemble (b) Dynamic ensemble

Fig. 15. Best predicted day—actual and predicted values .

Acknowledgements

The authors would like to thank the Spanish Ministry of Economy and Competitiveness and Junta de Andalucía for the support under projects TIN2017-8888209C2-1-R, TIN2014-55894-C2-R and P12-TIC-1728, respectively. Additionally, the authors want to express their gratitude to the T-Systems Iberia company since all experiments were carried out on its Open Telekom Cloud Platform based on the Open-Stack open source.

References

[1] M. Ge, H. Bangui, B. Buhnova, Big data for internet of things: A survey, Future Gener. Comput. Syst. (2018).
[2] X. Liu, P.S. Nielsen, A hybrid ICT-solution for smart meter data analytics, Energy 115 (2016) 1710–1722.
[3] Y. Sakurai, Y. Matsubara, C. Faloutsos, Mining and forecasting of big time-series data, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2015, pp. 919–922.
[4] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters.
[5] T. White, Hadoop, the Definitive Guide, O' Really Media, 2012.
[6] M. Hamstra, H. Karau, M. Zaharia, A. Knwinski, P. Wendell, Learning Spark: Lightning-Fast Big Analytics, O' Really Media, 2015.
[7] Machine learning library (mllib) for apache spark, 2016, On-line. http://spark.apache.org/docs/latest/mllib-guide.html.
[8] F. Hu, C. Yang, J.L. Schnase, D.Q. Duffy, M. Xu, M.K. Bowen, T. Lee, W. Song, ClimateSpark: An in-memory distributed computing framework for big climate data analytics, Comput. Geosci. 115 (2018) 154–166.
[9] M.E. Shafiee, Z. Barker, A. Rasekh, Enhancing water system models by integrating big data, Sustai. Cities Soc. 37 (2018) 485–491.
[10] S. Magana-Zook, J.M. Gaylord, D.R. Knapp, D.A. Dodge, S.D. Ruppert, Large-scale seismic waveform quality metric calculation using Hadoop, Comput. Geosci. 94 (2016) 18–30.
[11] D. Gomes-Mestre, C.E. Santos-Pires, D.C. Nascimento, A.R. Monteiro-de Queiroz, V. Borges-Santos, T. Brasileiro-Araujo, An efficient spark-based adaptive windowing for entity matching, J. Syst. Softw. 128 (2017) 1–10.
[12] T. Cerquitelli, Predicting large scale fine grain energy consumption, Energy Proc. 111 (2017) 1079–1088, 8th International Conference on Sustainability in Energy and Buildings, SEB-16, 11-13 2016, Turin, Italy.
[13] X. Zhang, U. Khanal, X. Zhao, S. Ficklin, Making sense of performance in in-memory computing frameworks for scientific data analysis: A case study of the spark system, J. Parallel Distrib. Comput. (2017).
[14] S. Caíno-Lores, A. Lapin, J. Carretero, P. Kropf, Applying big data paradigms to a large scale scientific workflow: Lessons learned and future directions, Future Gener. Comput. Syst. (2018).
[15] A. Galicia, J.F. Torres, F. Martínez-Álvarez, A. Troncoso, Scalable forecasting techniques applied to big electricity time series, in: Proceedings of the 14th International Work-Conference on Artificial Neural Networks, 2017, pp. 165–175.
[16] F. Martínez-Álvarez, A. Troncoso, G. Asencio-Cortés, J.C. Riquelme, A survey on data mining techniques applied to electricity-related time series forecasting, Energies 8 (11) (2015) 13162–13193.
[17] G. Box, G. Jenkins, Time Series Analysis: Forecasting and Control, John Wiley and Sons, 2008.
[18] C.W. Tsai, C.F. Lai, H.C. Chao, A. Vasilakos, Big data analytics: a survey, J. Big Data 2 (1) (2015) 21.
[19] L. Zhou, S. Pan, J. Wang, A.V. Vasilakos, Machine learning on big data: opportunities and challenges, Neurocomputing 237 (2017) 350–361.
[20] G. Cavallaro, M. Riedel, M. Richerzhagen, J.A. Benediktsson, A. Plaza, On understanding big data impacts in remotely sensed image classification using support vector machine methods, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 8 (2015) 4634–4646.
[21] J.L. Reyes-Ortiz, L. Oneto, D. Anguita, Big data analytics in the cloud: Spark on hadoop vs MPI/OpenMP on Beowulf, Proc. Comput. Sci. 53 (2015) 121–130.
[22] R. Talavera-Llames, R. Pérez-Chacón, M. Martínez-Ballesteros, A. Troncoso, F. Martínez-Álvarez, A nearest neighbours-based algorithm for big time series data forecasting, in: Proceedings of the International Conference on Hybrid Artificial Intelligence Systems, 2016, pp. 174–185.
[23] J. González-López, S. Ventura, A. Cano, Distributed nearest neighbor classification for large-scale multi-label data on spark, Future Gener. Comput. Syst. 87 (2018) 66–82.
[24] I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera, MRPR: A MapReduce solution for prototype reduction in big data classification, Neurocomputing 150 (2015) 331–345.
[25] N. Mehdiyev, J. Lahann, A. Emrich, D. Enke, P. Fettke, P. Loos, Time series classification using deep learning for process planning: A case from the process industry, Proc. Comput. Sci. 114 (2017) 242–249.
[26] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, A.Y. Zomaya, I. Khalil, F. Sebti, A. Bouras, A.Y. Zomaya, S. Foufou, A. Bouras, A survey of clustering algorithms for big data: Taxonomy & empirical analysis, IEEE Trans. Emerg. Top. Comput. 5 (3) (2014) 267–279.
[27] H. Asri, H. Mousannif, H.A. Moatassime, Real-time miscarriage prediction with SPARK, Proc. Comput. Sci. 113 (2017) 423–428.
[28] R. Pérez-Chacón, R. Talavera-Llames, F. Martínez-Álvarez, A. Troncoso, Finding electric energy consumption patterns in big time series data, in: Proceedings of the International Conference on Distributed Computing and Artificial Intelligence, 2016, pp. 231–238.
[29] J.M. Luna-Romera, M.M. Martínez-Ballesteros, J. García-Gutiérrez, J.C. Riquelme-Santos, J.C. Riquelme, M.M. Martínez-Ballesteros, J.C. Riquelme, An approach to silhouette and dunn clustering indices applied to big data in spark, in: Progress in Artificial Intelligence, Springer, Cham, 2016, pp. 160–169.
[30] H. Hassani, E.S. Silva, Forecasting with big data: A review, Ann. Data Sci. 1 (2) (2015) 5–19.
[31] G. Asencio-Cortés, A. Morales-Esteban, X. Shang, F. Martínez-Álvarez, Earthquake prediction in California using regression algorithms and cloud-based big data infrastructure, Comput. Geosci. 115 (2018) 198–210.
[32] J.F. Torres, A.M. Fernández, A. Troncoso, F. Martínez-Álvarez, Deep Learning-Based Approach for Time Series Forecasting with Application to Electricity Load, Springer International Publishing, 2017, pp. 203–212.
[33] I. Rodríguez-Fdez, M. Mucientes, A. Bugarín, S-FRULER: Scalable fuzzy rule learning through evolution for regression, Knowl.-Based Syst. 110 (2016) 255–266.
[34] L.I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, John Wiley & Sons, 2014.
[35] R. Polikar, Ensemble Learning, Springer US, 2012, pp. 1–34.
[36] B. Krawczyk, L.L. Minku, J. Gama, J. Stefanowski, Michał Woźniak, Ensemble learning for data stream analysis: A survey, Inf. Fusion 37 (2017) 132–156.
[37] L. Li, S. Bagheri, H. Goote, A. Hasan, G. Hazard, Risk adjustement of patient expenditures: A big data analytics approach, in: Proceedings of the IEEE International Conference on Big Data, 2013, pp. 12–14.
[38] B. Panda, J.S. Herbach, S. Basu, R.J. Bayardo, PLANET: massively parallel learning of tree ensembles with MapReduce, in: Proceedings of the Very Large Databases, 2009, pp. 1426–1437.
[39] K. Singh, S.C. Guntuku, A. Thakur, C. Hota, Big data analytics framework for peer-to-peer botnet detection using random forests, Inform. Sci. 278 (2014) 488–497.
[40] J.H. Abawajy, A. Kelarev, M. Chowdhury, Large iterative multitier ensemble classifiers for security of big data, IEEE Trans. Emerg. Top. Comput. 2 (3) (2014) 352–363.
[41] Y. Yan, Q. Zhu, M.L. Shyu, S.C. Chen, A classifier ensemble framework for multimedia big data classification, in: 2016 IEEE 17th International Conference on Information Reuse and Integration (IRI), jul 2016, pp. 615–622.
[42] T. Do, F. Poulet, Random local SVMs for classifying large datasets, in: Proceedings of the International Conference on Future Data and Security Engineering, 2015, pp. 3–15.
[43] V. Estruch, C. Ferri, J. Hernández-Orallo, M.J. Ramírez-Quintana, Shared ensemble learning using multi-trees, in: Proceedings of the 8th Ibero-American Conference on Artificial Intelligence, 2002, pp. 204–213.
[44] A. Torres-Barrán, Á. Alonso, J.R. Dorronsoro, Regression tree ensembles for wind energy and solar radiation prediction, Neurocomputing (2017).
[45] M.A. Hassan, A. Khalil, S. Kaseb, M.A. Kassem, Exploring the potential of tree-based ensemble methods in solar radiation modeling, Appl. Energy 203 (Suppl. C) (2017) 897–916.
[46] J.F. Torres, A. Troncoso, I. Koprinska, Z. Wang, F. Martínez-Álvarez, Deep learning for big data time series forecasting applied to solar power, in: International Joint Conference SOCO'18-CISIS'18-ICEUTE'18, Springer International Publishing, 2019, pp. 123–133.
[47] Z. Wang, I. Koprinska, M. Rana, Solar power forecasting using pattern sequences, in: Artificial Neural Networks and Machine Learning – ICANN 2017, Springer International Publishing, 2017, pp. 486–494.
[48] F. Martínez-Álvarez, A. Troncoso, J.C. Riquelme, J.S. Aguilar Ruiz, Energy time series forecasting based on pattern sequence similarity, IEEE Trans. Knowl. Data Eng. 23 (8) (2011) 1230–1243.

Bibliografía

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, pages 207–216. ACM, 1993.

[2] J. Alcalá-Fdez, L. Sánchez, S. Garcia, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, et al. Keel: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2009.

[3] A. Alexandrov, R. Bergmann, S. Ewen, J. C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, et al. The stratosphere platform for big data analytics. *The VLDB Journal The International Journal on Very Large Data Bases*, 23(6):939–964, 2014.

[4] S. Almon. The distributed lag between capital appropriations and expenditures. *Econometrica: Journal of the Econometric Society*, pages 178–196, 1965.

[5] M. A. Alsheikh, D. Niyato, S. Lin, H. Tan, and Z. Han. Mobile big data analytics using deep learning and apache spark. *IEEE network*, 30(3):22–29, 2016.

[6] G. Asencio-Cortés, E. Florido, A. Troncoso, and F. Martínez-Álvarez. A novel methodology to predict urban traffic congestion with ensemble learning. *Soft Computing*, 20(11):4205–4216, 2016.

[7] K. Ashton et al. That "internet of things" thing. *RFID journal*, 22(7):97–114, 2009.

[8] T. O. Ayodele. Introduction to machine learning. In *New Advances in Machine Learning*. InTech, 2010.

[9] F. Bach, H. K. Çakmak, H. Maass, and U. Kuehnapfel. Power grid time series data analysis with pig on a hadoop cluster compared to multi core systems. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 208–212, Feb 2013.

[10] A. Bassi and G. Horn. Internet of things in 2020: A roadmap for the future. *European Commission: Information Society and Media*, 22:97–114, 2008.

[11] G. E. Box and G. M. Jenkins. *Time series analysis: forecasting and control, revised ed.* Holden-Day, 1976.

[12] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis.* Wiley, 2013.

[13] G. E. P. Box and G. C. Tiao. Intervention analysis with applications to economic and environmental problems. *Journal of the American Statistical association*, 70(349):70–79, 1975.

[14] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[15] P. J. Brockwell and R. A. Davis. *Time series: theory and methods.* Springer Science & Business Media, 2013.

[16] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[17] K. Chakraborty, K. Mehrotra, C. K. Mohan, and S. Ranka. Forecasting the behavior of multivariate time series using neural networks. *Neural Networks*, 5(6):961–970, 1992.

[18] Open Telekom Cloud. On-line. *https://cloud.telekom.de/en*, 2018.

[19] D. Cook. *Practical machine learning with H2O: powerful, scalable techniques for deep learning and AI.* O'Reilly Media, Inc., 2016.

[20] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[21] R. F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica: Journal of the Econometric Society*, pages 987–1007, 1982.

[22] R. F. Engle and C. J. Granger. Co-integration and error correction: representation, estimation, and testing. *Econometrica: journal of the Econometric Society*, pages 251–276, 1987.

[23] V. Estivill-Castro. Why so many clustering algorithms: a position paper. *ACM SIGKDD explorations newsletter*, 4(1):65–75, 2002.

[24] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. Advances in knowledge discovery and data mining. 1996.

[25] D. F. Findley, B. C. Monsell, W. R. Bell, M. C. Otto, and B. Chen. New capabilities and methods of the X-12-ARIMA seasonal-adjustment program. *Journal of Business & Economic Statistics*, 16(2):127–152, 1998.

[26] GraphX: Apache Spark's API for graphs and graph-parallel computation. On-line. *https://spark.apache.org/docs/latest/graphx-programming-guide.html*, 2018.

[27] Machine Learning Library (MLlib) for Spark. On-line. *http://spark.apache.org/docs/latest/mllib-guide.html*, 2018.

[28] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, and F. Martínez-Álvarez. Multi-step forecasting for big data time series based on ensemble learning. *Knowledge-Based Systems*, 163:830–841, 2019.

[29] A. Galicia, J. F. Torres, F. Martínez-Álvarez, and A. Troncoso. Scalable forecasting techniques applied to big electricity time series. In *Proceedings of the 14th International Work-Conference on Artificial Neural Networks (IWANN)*, pages 165–175, 2017.

[30] A. Galicia, J. F. Torres, F. Martínez-Álvarez, and A. Troncoso. A novel spark-based multi-step forecasting algorithm for big data time series. *Information Sciences*, 467:800–818, 2018.

[31] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. *ACM SIGOPS Operating System Review*, 37(5):29–43, 2003.

[32] A. Gounaris and J. Torres. A methodology for spark parameter tuning. *Big data research*, 11:22–32, 2018.

[33] C. J. Granger. Developments in the study of cointegrated economic variables. *Oxford Bulletin of economics and statistics*, 48(3):213–228, 1986.

[34] C. J. Granger and P. Newbold. Spurious regressions in econometrics. *Journal of econometrics*, 2(2):111–120, 1974.

[35] P. Guillemin, P. Friess, et al. Internet of things strategic research roadmap. *The Cluster of European Research Projects, Tech. Rep*, 2009.

[36] M. Gupta, A. B. Sharma, H. Chen, and G. Jiang. Context-aware time series anomaly detection for complex systems. In *Workshop Notes*, 2013.

[37] W. K. Härdle and L. Simar. Canonical correlation analysis. In *Applied multivariate statistical analysis*, pages 443–454. Springer, 2015.

[38] H. Hassani and E. S. Silva. Forecasting with big data: A review. *Annals of Data Science*, 2(1):5–19, 2015.

[39] J. Hernández Orallo, C. Ferri Ramirez, and M. J. Ramirez Quintana. *Introducción a la Minería de Datos*. Pearson Prentice Hall, 2004.

[40] T. Hill, L. Marquez, M. O'Connor, and W. Remus. Artificial neural network models for forecasting and decision making. *International journal of forecasting*, 10(1):5–15, 1994.

[41] H. S. Hippert, C. E. Pedreira, and R. C. Souza. Neural networks for short-term load forecasting: a review and evaluation. *IEEE Transactions on Power Systems*, 16(1):44–55, Feb 2001.

[42] M. J. Hu. *Application of the adaline system to weather forecasting*. PhD thesis, Department of Electrical Engineering, Stanford University, 1964.

[43] L. M. Koyck. *Distributed Lags and Investment Analysis*. Contributions to Economic Analysis, 4. 1954.

[44] A. Lapedes and R. Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. Technical report, 1987.

[45] N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1939–1947. ACM, 2015.

[46] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[47] J. Leibiusky, G. Eisbruch, and D. Simonassi. *Getting Started with Storm*. O'Reilly Media, Inc., 2012.

[48] B. Liu, J. Li, C. Chen, W. Tan, Q. Chen, and M. Zhou. Efficient motif discovery for large-scale time series in healthcare. *IEEE Transactions on Industrial Informatics*, 11(3):583–590, 2015.

[49] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pages 281–297. Oakland, CA, USA, 1967.

[50] F. Martínez-Álvarez, A. Troncoso, J. C. Riquelme, and J. S. Aguilar Ruiz. Energy time series forecasting based on pattern sequence similarity. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1230–1243, 2011.

[51] N. Marz and J. Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications Co., 2015.

[52] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In *Advances in neural information processing systems*, pages 512–518, 2000.

[53] A. De Mauro, M. Greco, and M. Grimaldi. A formal definition of big data based on its essential features. *Library Review*, 65(3):122–135, 2016.

[54] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[55] G. McLachlan. *Discriminant analysis and statistical pattern recognition*. John Wiley & Sons, 2004.

[56] K. Mirko and J. W. Kantelhardt. Hadoop.TS: large-scale time-series processing. *International Journal of Computer Applications*, 74(17), 2013.

[57] K. Nikolopoulos and F. Petropoulos. Forecasting for big data: Does suboptimality matter? *Computers & Operations Research*, 98:322 – 329, 2018.

[58] Spark officially sets a new record in large-scale sorting. On-line. *http://databricks.com/blog/2014/11/05/sparkofficially-sets-a-new-record-in-large-scale-sorting.html*, 2014.

[59] R package (neuralnet). On-line. *https://CRAN.R-project.org/package=neuralnet*, 2016.

[60] R package (randomForestSRC). On-line. *https://CRAN.R-project.org/package=randomForestSRC*, 2017.

[61] Sort Benchmark Home Page. On-line. *http://sortbenchmark.org/*, 2014.

[62] R. Pérez-Chacón, J. M. Luna-Romera, A. Troncoso, F. Martínez-Álvarez, and J. C. Riquelme. Big data analytics for discovering electricity consumption patterns in Smart Cities. *Energies*, 11(3), 2018.

[63] R. Pérez-Chacón, R. L. Talavera-Llames, F. Martínez-Álvarez, and A. Troncoso. Finding electric energy consumption patterns in big time series data. In *Proceedings of the 13th International Conference on Distributed Computing and Artificial Intelligence*, pages 231–238, 2016.

[64] C. Pérez López. Econometría de las series temporales. Technical report, 2006.

[65] M. Peris-Ortiz, D. R. Bennett, and D. P. B. Yábar. *Sustainable Smart Cities: Creating Spaces for Technological, Social and Business Development*. Springer International Publishing, 2016.

[66] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[67] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[68] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhua, J. Zakaria, and E. Keogh. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Trans. Knowl. Discov. Data*, 7(3):10:1–10:31, September 2013.

[69] M. Rana, I. Koprinska, and V. G. Agelidis. Univariate and multivariate methods for very short-term solar photovoltaic power forecasting. *Energy Conversion and Management*, 121(Supplement C):380 – 390, 2016.

[70] L. Rokach and O. Z. Maimon. *Data mining with decision trees: theory and applications*. World scientific, 2008.

[71] E. J. Ruiz, V. Hristidis, C. Castillo, A. Gionis, and A. Jaimes. Correlating financial time series with micro-blogging activity. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 513–522. ACM, 2012.

[72] C. Salperwyck, S. Maby, J. Cubillé, and M. Lagacherie. Courbospark: Decision tree for time-series on spark. In *AALTD@ PKDD/ECML*, 2015.

[73] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[74] Amazon Web Services. On-line. *https://aws.amazon.com/es/*, 2018.

[75] J. Shieh and E. Keogh. iSAX: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631. ACM, 2008.

[76] J. Shiskin. *The X-11 variant of the census method II seasonal adjustment program*. US Government Printing Office, 1965.

[77] R. Shyam, B. G. HB, S. Kumar, P. Poornachandran, and K. P. Soman. Apache spark a big data analytics platform for smart grid. *Procedia Technology*, 21:171–178, 2015.

[78] C. A. Sims. Macroeconomics and reality. *Econometrica: Journal of the Econometric Society*, pages 1–48, 1980.

[79] Google Cloud Storage. On-line. *https://cloud.google.com/storage/*, 2018.

[80] International Data Corporation. Digital Universe Study. On-line. *http://www.emc.com/leadership/digital-universe/index.htm*, 2014.

[81] R. Talavera-Llames, R. Pérez-Chacón, A. Troncoso, and F. Martínez-Álvarez. Big data time series forecasting based on nearest neighbours distributed computing with spark. *Knowledge-Based Systems*, 161:12–25, 2018.

[82] R. Talavera-Llames, R. Pérez-Chacón, A. Troncoso, and F. Martínez-Álvarez. MV-kWNN: A novel multivariate and multi-output weighted nearest neighbours algorithm for big data time series forecasting. *Neurocomputing*, in press, 2018.

[83] R. L. Talavera-Llames, R. Pérez-Chacón, M. Martínez-Ballesteros, A. Troncoso, and F. Martínez-Álvarez. A nearest neighbours-based algorithm for big time series data forecasting. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 174–185. Springer, 2016.

[84] Z. Tang, C. De Almeida, and P. A. Fishwick. Time series forecasting using neural networks vs. Box-Jenkins methodology. *Simulation*, 57(5):303–310, 1991.

[85] J. F. Torres, A. M. Fernández, A. Troncoso, and F. Martínez-Álvarez. Deep learning-based approach for time series forecasting with application to electricity load. In *Proceedings of the International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC)*, pages 203–212, 2017.

[86] J. F. Torres, A. Galicia, A. Troncoso, and F. Martínez-Álvarez. A scalable approach based on deep learning for big data time series forecasting. *Integrated Computer-Aided Engineering*, (Preprint):1–14, 2018.

[87] J. F. Torres, A. Troncoso, I. Koprinska, Z. Wang, and F. Martínez-Álvarez. Deep learning for big data time series forecasting applied to solar power. In *The 13th International Conference on Soft Computing Models in Industrial and Environmental Applications*, pages 123–133. Springer, 2018.

[88] A. Troncoso, J. M. Riquelme-Santos, A. Gómez-Expósito, J. L. Martínez-Ramos, and J. C. Riquelme. Electricity market price forecasting based on weighted nearest neighbors techniques. *IEEE Transactions on Power Systems*, 22(3):1294–1301, 2007.

[89] R. S. Tsay. *Multivariate time series analysis: with R and financial applications*. John Wiley & Sons, 2013.

[90] E. Uriel and A. Peiro. *Introducción al análisis de series temporales*. Paraninfo, 2000.

[91] Y. Wang and G. Li. An efficient data aggregation scheme in wireless sensor networks. In *Internet of Things*, pages 25–32. Springer Berlin Heidelberg, 2012.

[92] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting sunspots and exchange rates with connectionist networks. Technical report, PRE-33772, 1992.

[93] P. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

[94] I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, and S. J. Cunningham. Weka: Practical machine learning tools and techniques with java implementations. 1999.

[95] C. Xiaojun, L. Xianpeng, and X. Peng. IOT-based air pollution monitoring and forecasting system. In *2015 International Conference on Computer and Computational Sciences (ICCCS)*, pages 257–260, 2015.

[96] X. Yan and X. Su. *Linear regression analysis: theory and computing*. World Scientific, 2009.

[97] A. Zagorecki. A versatile approach to classification of multivariate time series data. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 407–410, Sept 2015.

[98] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 15–28, 2012.

Tienes que creer en ti mismo.

El arte de la guerra.
Sun Tzu

Todos los finales son también comienzos,
lo que pasa es que no lo sabemos en su momento.

Mitch Alborn